Machine Learning I

Lecture 1 - Introduction to Machine Learning

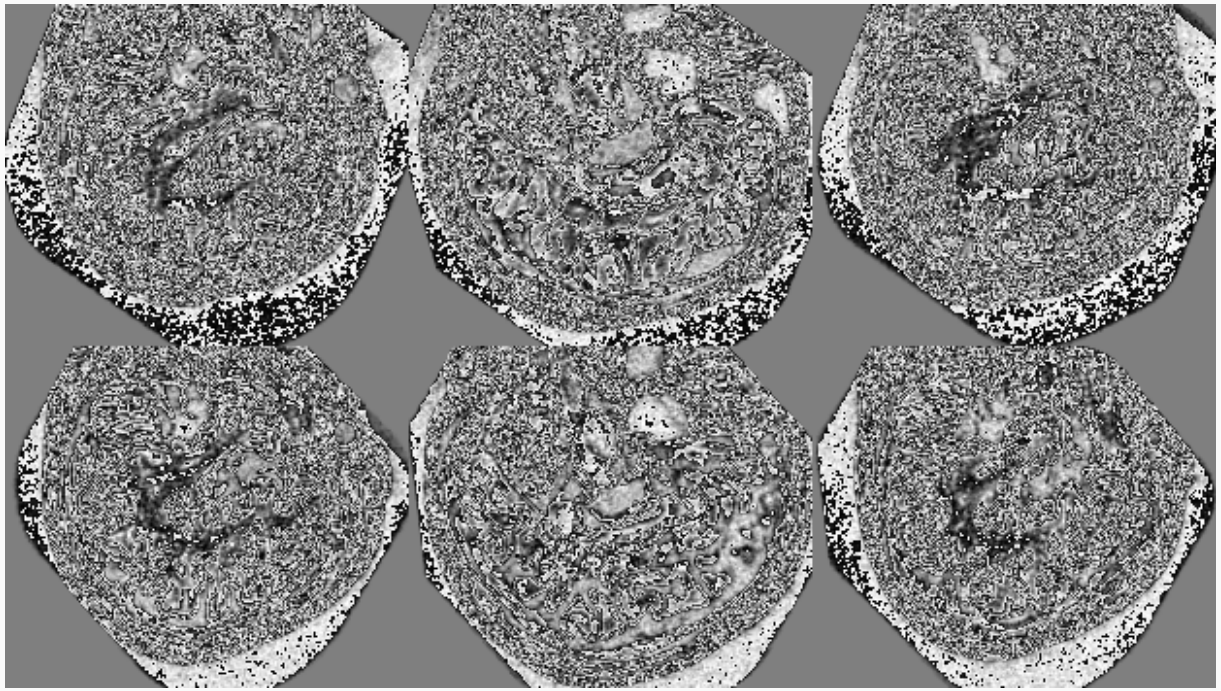
Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

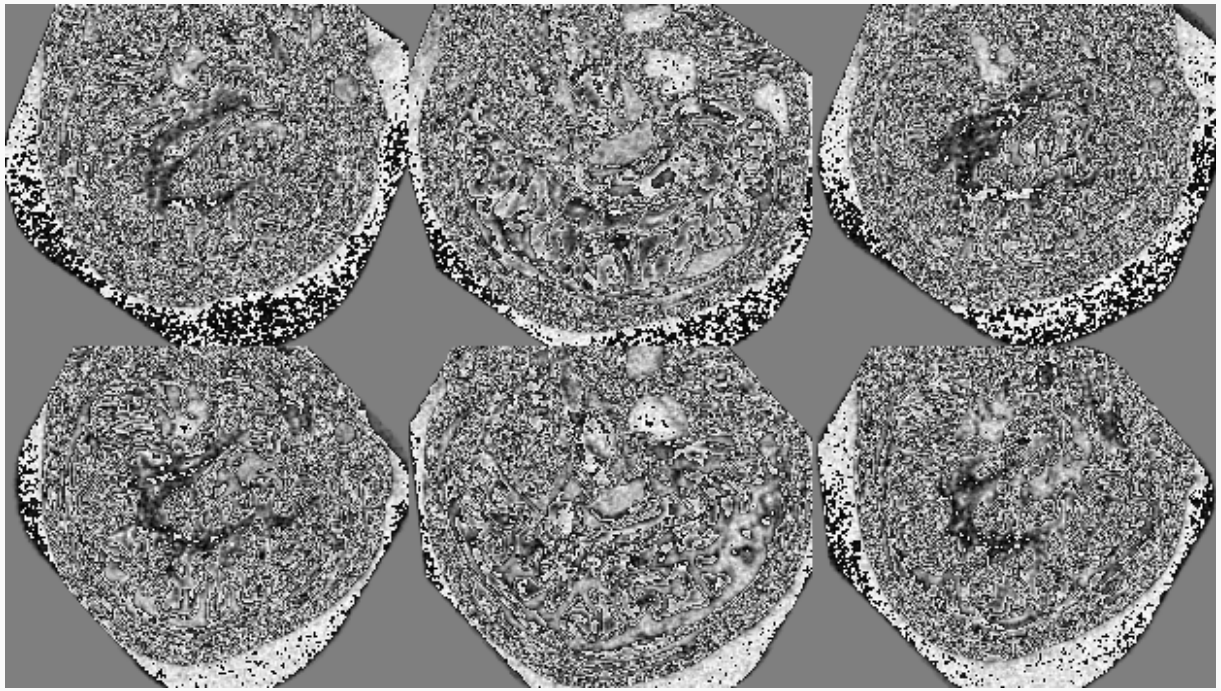
What is Machine Learning?

Statistical Modeling



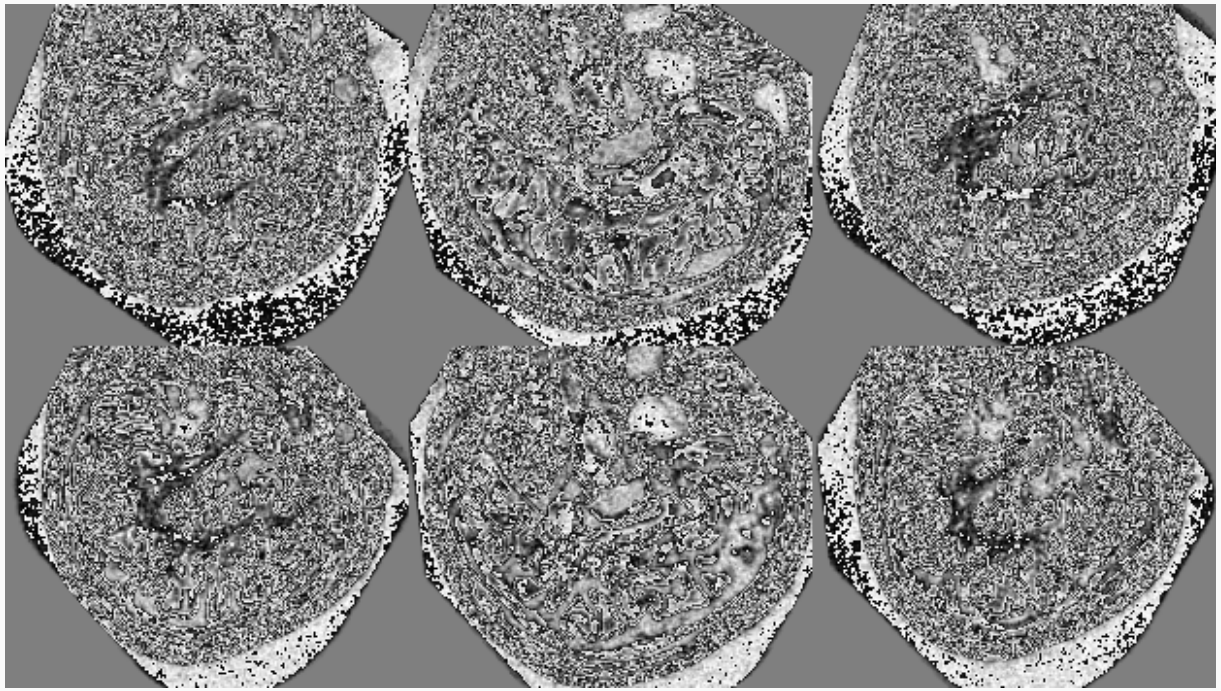
In the MRI scans above (from the OASIS 1 data set), the top three images come from a patient who has not been diagnosed with dementia. The bottom image come from a patient who has been diagnosed with severe dementia. Human doctors looking at these images wouldn't expect to be able to detect any difference between the brains, but maybe computational methods can do better?

Statistical Modeling



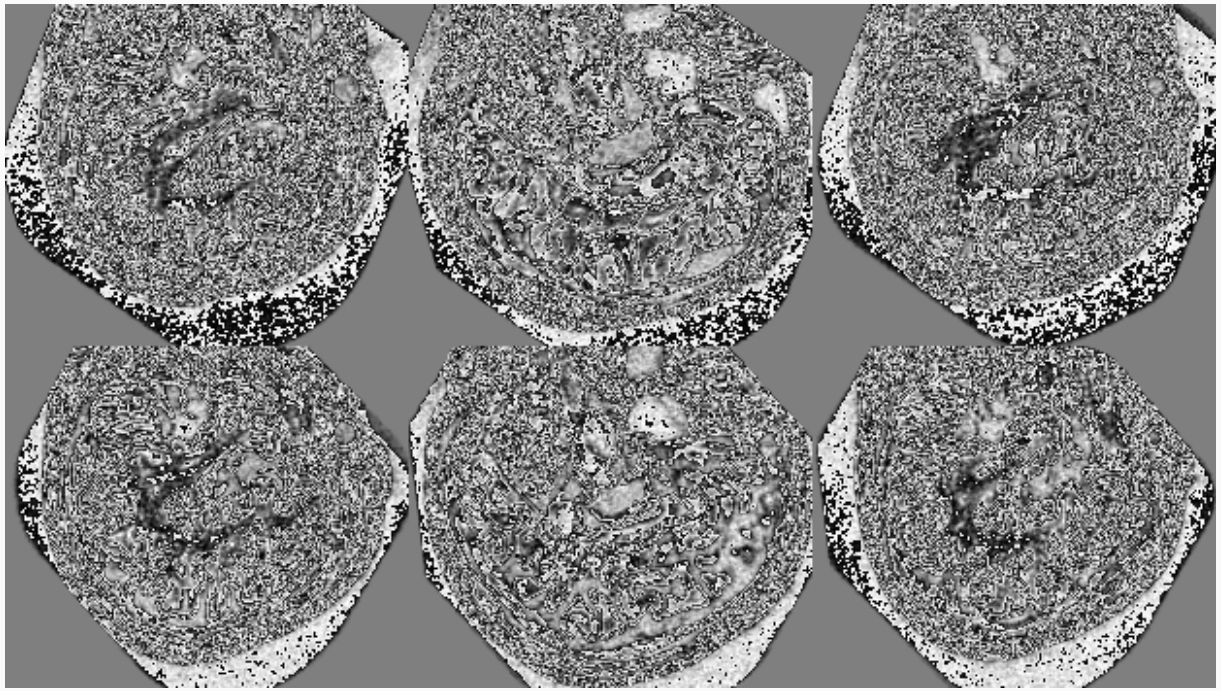
But how can we discover how this data is organized? Is dementia normally distributed? Each MRI is contains 6,443,008 pixels and so mathematically is an element of $\mathbb{R}^{6443008}$, can we hope to write down a distribution on this space?

Statistical Modeling



Machine learning is discipline that deals with fitting mathematical models to intractable distributions of data. In recent years, computational increases have enabled us to attack higher and higher dimensional problems, sometimes with very little data, sometimes with terabytes of information.

Statistical Modeling



In low dimensions and with sparse data, empirical reasoning often leads to careful and informative models. In the high dimensional world of machine learning, a suite of so called "black box" models has been developed that can be tuned to a wide variety of problems. Some of these models are straightforward, like linear regression, but some like neural networks may contain millions of variables, and it is still an open question as to what structure exactly they are probing.

Statistical Modeling

In this course, we will learn both the mathematical and the implementation of the core machine learning methods: Linear models, linear basis fitting, neural networks, decision trees and cluster analysis. Over the course of the semester we will analyze and implement all of the mathematical pieces that go into a machine learning algorithm, and show how they fit into the physical implementation of a machine learning project. Along the way, we will talk about how to combine and ensemble train models, boot strap data and use machine learning to learn which algorithm to use.

We will finish the semester by putting machine learning in it's proper mathematical context, discussing learnability, the partially approximately correct learning framework, the No Free Lunch Theorem, and VC Dimension.

Throughout the course, we will be working on a large scale machine learning project with the aim to bring all of our methods and methodologies to bear on solving hard, real world problems.

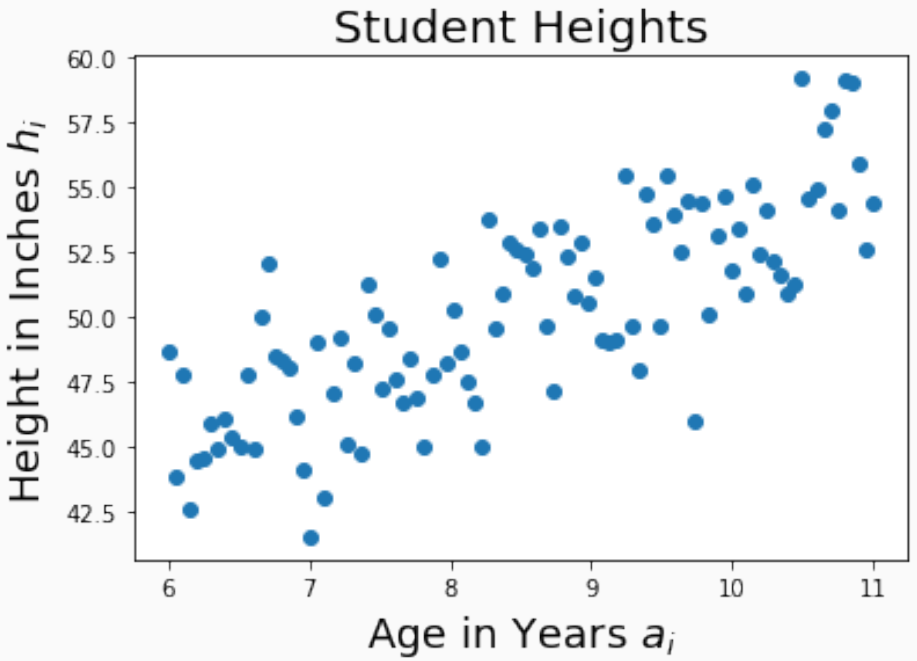
Statistical Learning Theory

Statistical Learning Theory

This course will use **statistical learning theory** to understand the mathematical aspects of **machine learning**. Machine learning is the process of fitting the parameters of a model to a given set of data.

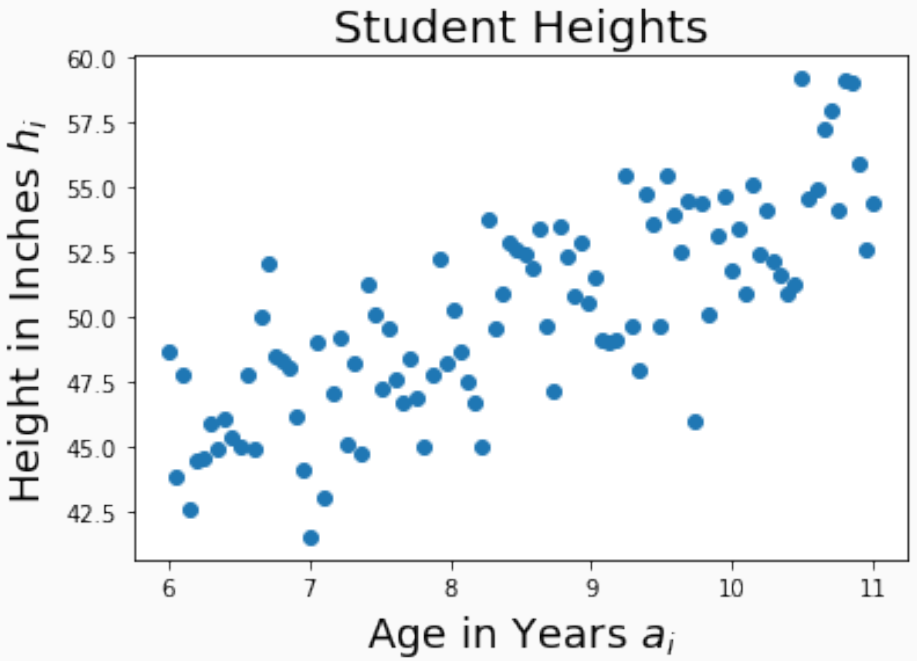
Statistical learning theory is roughly about understanding how much data is required to make predictions using a fit model to a certain degree of accuracy. We will first understand some theoretical aspects of the theory, and then move to practical implementation using Python.

Statistical Modeling



Example: Student height: In a school, students ages a_i are uniformly distributed between ages 6 and 11. We can fit a mathematical function to this data to model it, and then use the function to predict the heights of a new set of students using the data.

Statistical Modeling

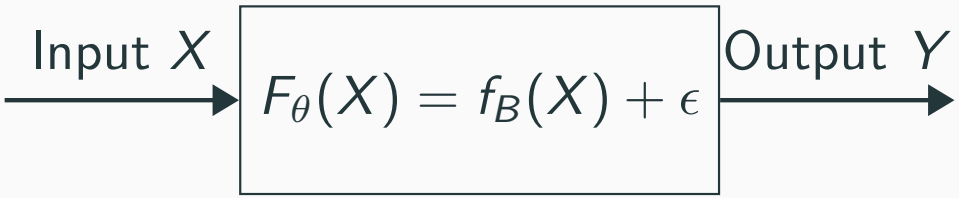


There are many choices that go into such a modeling project: Should we use a **linear** model or a **polynomial** model? For a linear model $h_i = c\,a_i + d$, how do we pick c and d based on the data? Once we fit the data, where is it reasonable to use the model? How much data do we need to get an accurate answer?

Statistical Modeling

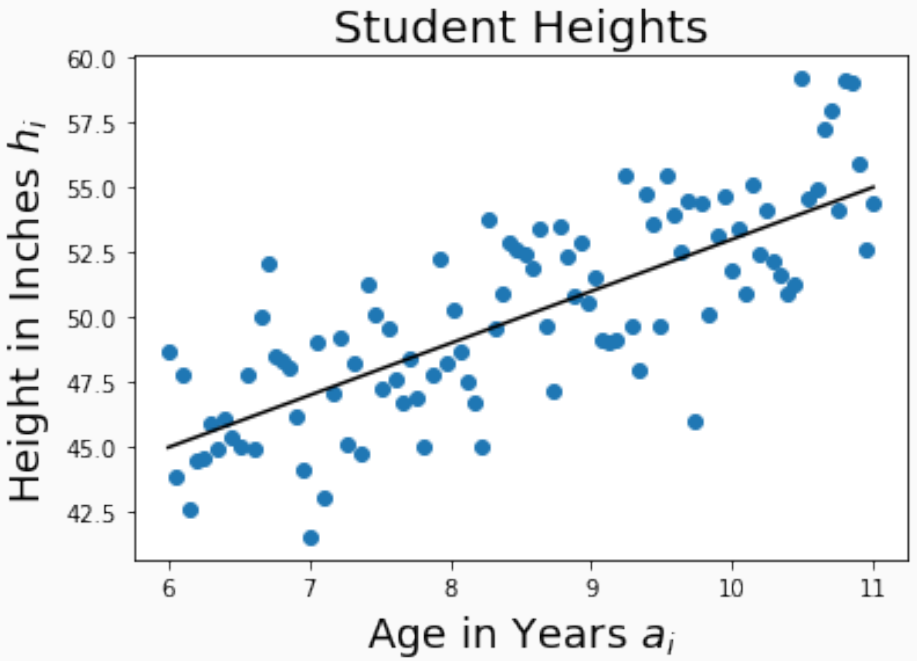
Formally, **Statistical modeling** starts with a sample space S, with data points distributed by an (unknown) probability distribution on S. We then posit a mathematical model $F_\theta(X)$ depending on parameters θ to generate estimates about the likelihood of observed data.

In the previous example, if $X \in \mathbb{R}$ is age and $Y \in \mathbb{R}$ is height, and we fit some function $f_B : X \to Y$ to give a labeling of data. By adding in a random variable ϵ with variance σ^2, we can write the statistical model



We then pick θ to maximize the probability, or likelihood, the the datapoints we observe are drawn from the model. In the above, $\theta = (B, \sigma^2)$ parameterize both the choices in our model θ, and the choices about the noise σ^2.

Statistical Modeling

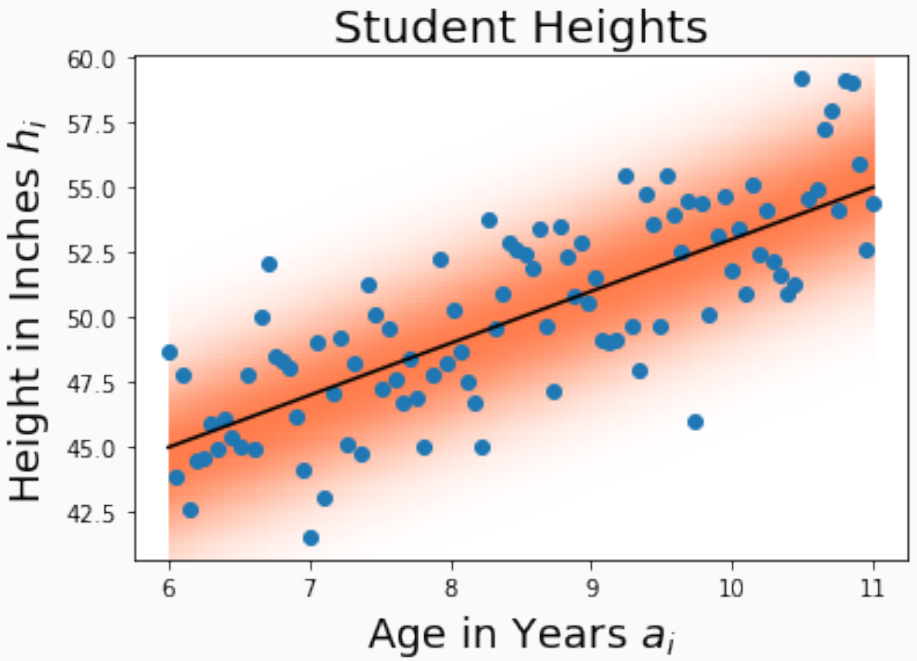


We might then predict height h_i of child i via their age a_i, using the model

$$h_i = b_0 + b_1 a_i + \epsilon_i \, .$$

We include a stochastic variable ϵ_i since h_i must fit all the data. To do statistical inference, we make the assumption that ϵ_i is Gaussian.

Statistical Modeling

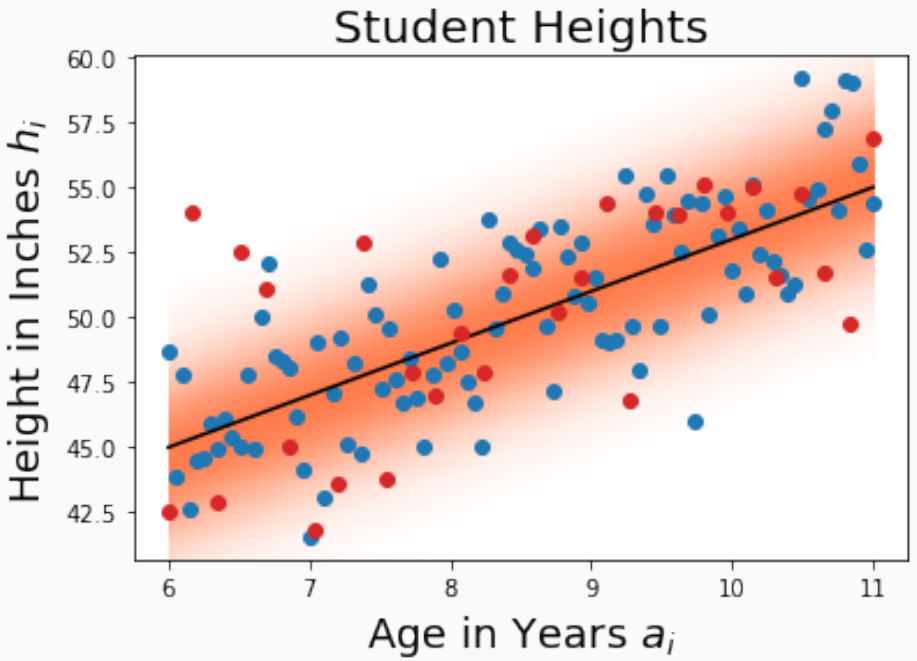


Model : $h_i = b_0 + b_1 a_i + \epsilon_i$

Sample space: $S = [6, 11] \times \mathbb{R}^+$ all possible pairs (a, h).

Parameters: $\theta = (\sigma^2, b_0, b_1)$. Each triple of values defines a model distribution P_θ on X. P is the set of all such distributions.

Statistical Modeling



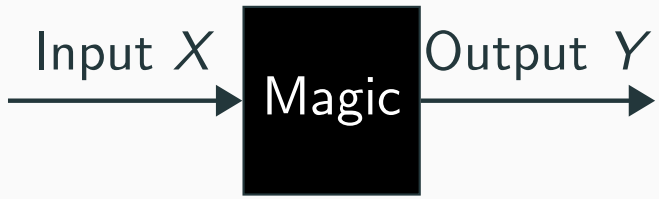
Model : $h_i = b_0 + b_1 a_i + \epsilon_i$

Sample space: $X = [6, 11] \times \mathbb{R}^+$ all possible pairs (a, h).

Parameters: $\theta = (\sigma_\epsilon^2, b_0, b_1)$. Each triple of values defines a distribution P_θ on X. P is the set of all such distributions.

Machine Learning vs Statistics

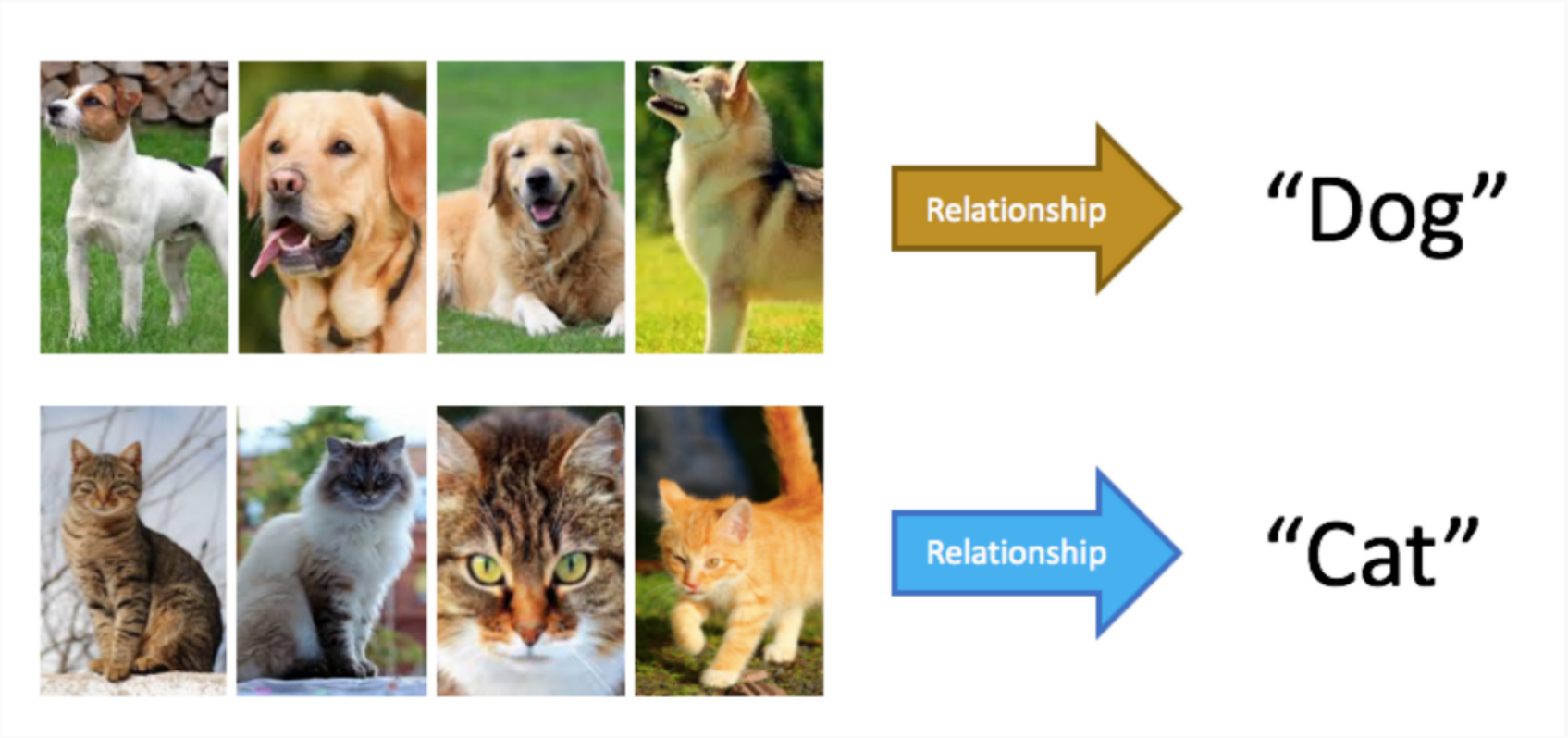
Machine learning is used in problems where the underlying distributions are unknown or unknowable. Typical examples include image classification, voice recognition, or gene expression.



Machine learning uses black-box methods that strive for accuracy regardless of the underlying distribution.

Main Question: For a given class of models, how many data points do we need in order to be certain (probabilistically) that the error in our model is small *no matter what the underlying distribution*.

Machine Learning Examples: Cats and Dogs



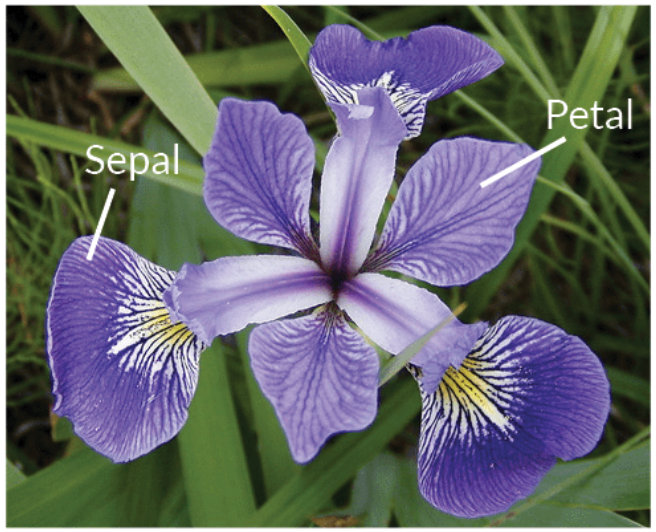
Example: In image classification, input $X = \mathbb{R}^{3\times 100\times 200}$ and output $Y = \{\text{"Dog"}, \text{"Cat"}\}$.

How do we find the distribution on the state space $S = X \times Y$?

Machine Learning Examples: Computer Vision

Example: For multilable classification, input $X = \mathbb{R}^{20\times 20}$ and output $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Above is the famous MNIST data set of handwritten numbers.

Machine Learning Examples: Iris Classification



Iris Versicolor

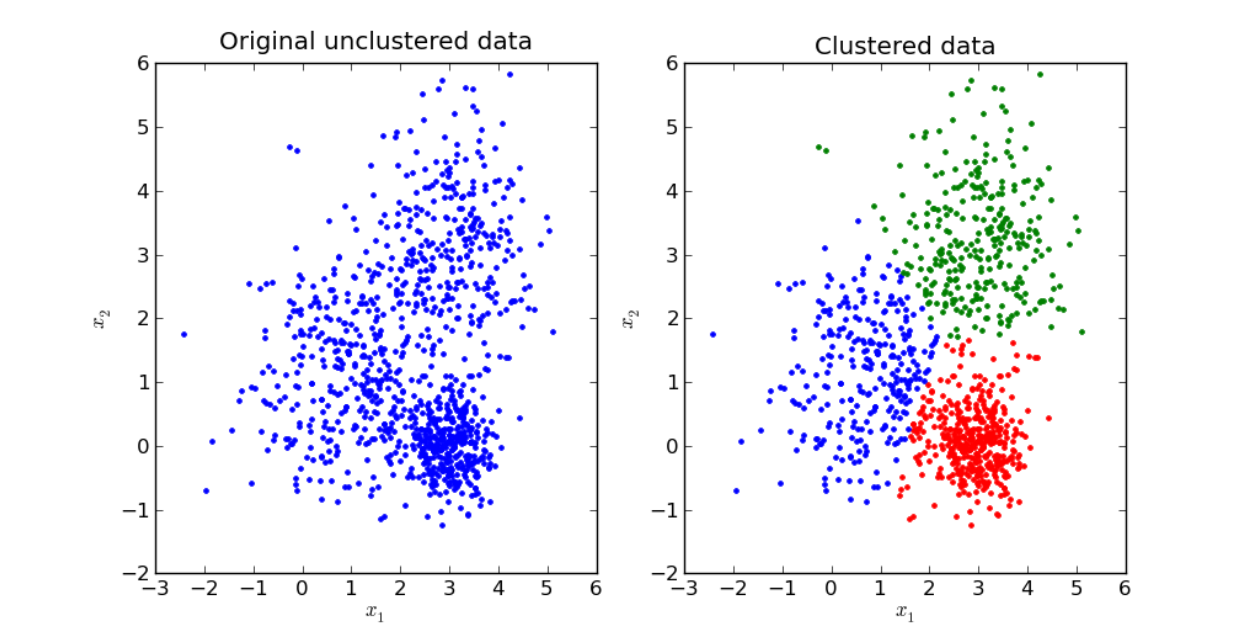
Iris Setosa

Iris Virginica

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	I. Setosa
7.0	3.2	4.7	1.4	I. Versicolor
		⋮		

Example: Classification by Attributes: Input is $X = \mathbb{R}^4$ and output is a finite set $\mathcal{G} = \{\text{Virginica, Setosa, Versicolor}\}$.

Machine Learning Examples: Market Segmentation

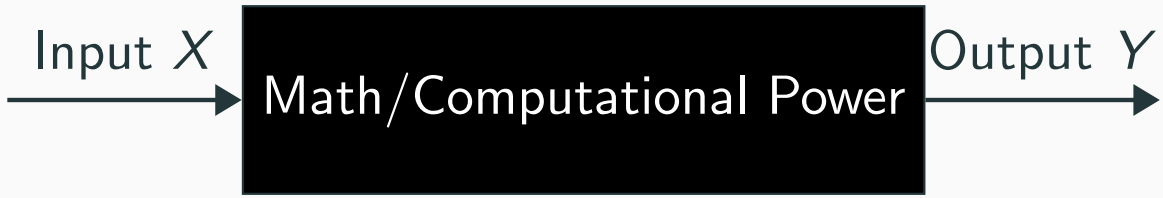


UserID	Products
0x131432	{"Leather Hat", "Dvvorsh Black Belt", "Dukes Hat Polish",...}
0x152341	{"Tinga XL Beach Towel", "Speco Meat Thermometer",...}

Example: Clustering: Input is $X = \mathbb{P}(\{\text{All Products}\})$, the powerset of the set of all products.

Machine Learning vs Statistics

Machine learning is used in problems where the underlying distributions are unknown or unknowable.



Main Question: For a given class of models, how many data points do we need in order to be certain (probabilistically) that the error in our model is small *no matter what the underlying distribution*.

Terminology and Notation

Two Types of Machine Learning

Supervised Learning: Given a sample set of *labeled* data, can we predict the labels on new unlabeled data from the same domain?

Examples: Image classification, classification by features.

Unsupervised Learning: Given a set of *unlabeled* data, can we find structure within the data?

Examples: Clustering, dimensional reduction.

Variable Types

There are two large families of variable types:

Qualitative Variables: Variables that only take discrete values, usually in a set of descriptive **classes**. Also called **categorical** or **discrete** variables, or **factors**. There could be no additional ordering or structure on the classes.

Examples: Iris name, "Cat" or "Dog", items purchased, the integers 0 to 9 in MNIST.

Quantitative Variables: Variables that take quantitative values, with some values larger than others and close values designating similar characteristics. Also called **numerical**.

Examples: Body temperature, grayscale value of a pixel, stem length, height.

Regression

Regression analysis is a set of procedures for estimating the relationships between independent variables (inputs) and dependent variables (outputs).

In ESL, **regression** specifically refers to prediction when the outputs are quantitative. If the outputs are qualitative, this is referred to as **classification**.

In UML, **regression** refers to any *functional* relationship between the domain of the inputs X and the range of the outputs Y, regardless of variable type.

This ambiguity can be found throughout the field.

Inputs and Output

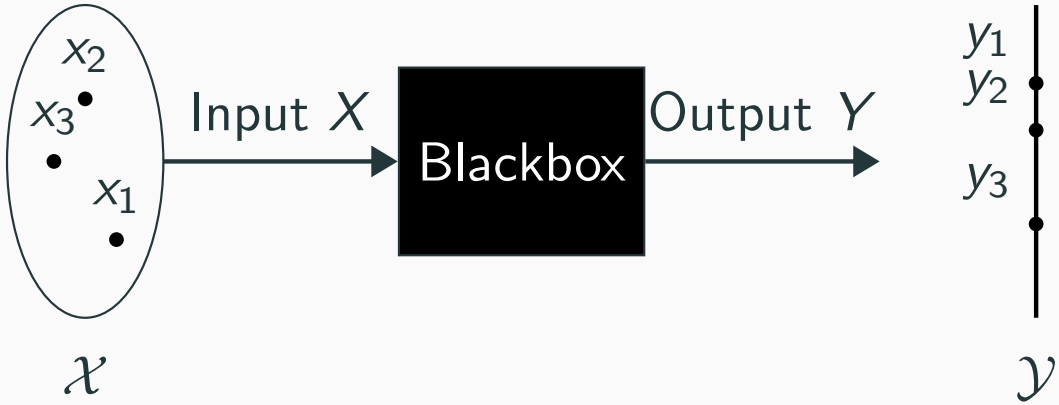
Input variables will be denoted by X. If X is a vector, its components will be X_j. The domain of these variables will be denoted $\mathcal{X}$.

In general, **output variables** will be denoted by Y. If we need to distinguish that the output is qualitative we will denote it by G. The domain of these variables will be $\mathcal{Y}$ and $\mathcal{G}$ respectively.

We use uppercase X, Y, G to refer to generic aspects of the variable and lowercase to denote observed values. For example, x_i denotes the i'th *observed* value of X and is a vector if X is.

All vectors are assumed to be column vectors.

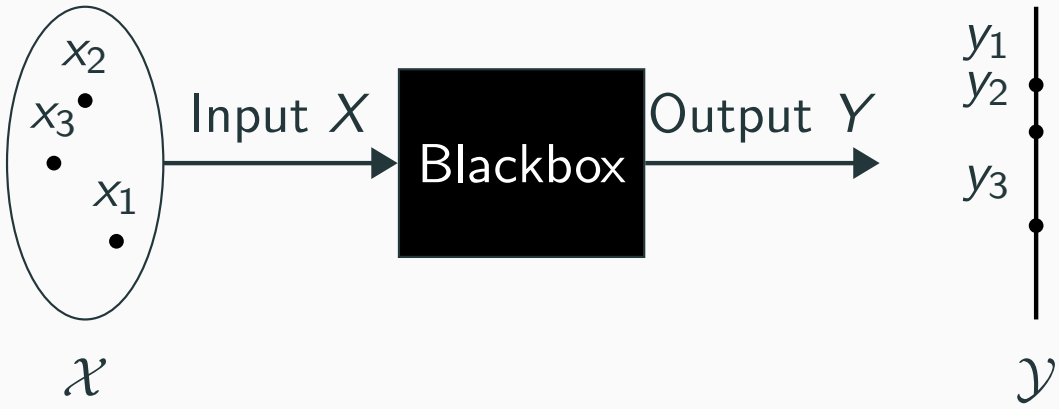
Inputs and Output



We assume that we have access to a set of N observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, for $i = 1, \ldots, N$. This set of observations is called the **training set** or **training data**.

N will almost always denote the number of observed points (data points in the training set).

Inputs and Output



Matrices will be represented by bold uppercase letters. For example, the set of N input p vectors x_i, $i = 1, \ldots, N$ will be represented as $\mathbf{X}$.

We will only bold a vector when it has N components. This is to distinguish the p vector of inputs x_i from the N vector $\mathbf{x}_j$ consisting of all observations on the variable X_j.

Since all vectors are assumed to be column vectors, the i'th row of $\mathbf{X}$ is x_i^T.

Example: Inputs and Outputs

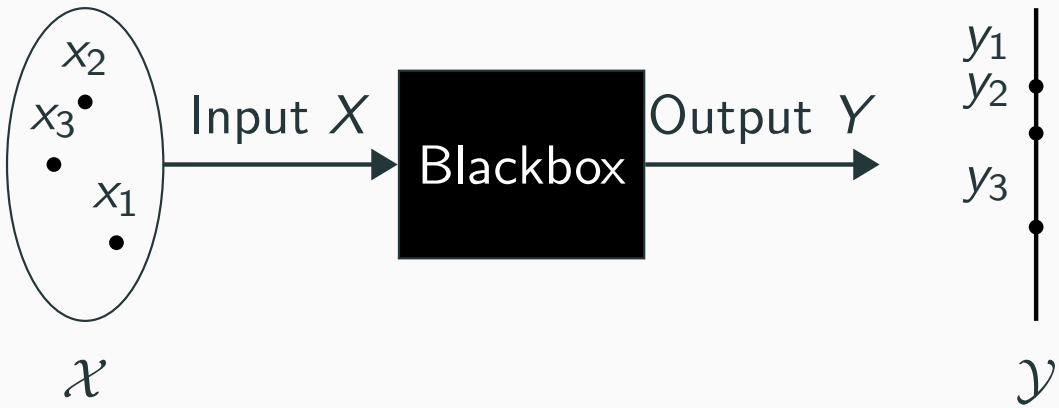
Sepal Length	Sepal Width	Petal Length	Petal Width
5.1	3.5	1.4	0.2
7.0	3.2	4.7	1.4
		$\vdots$	

For the domain of the Iris data,

$$x_1 = \begin{bmatrix} 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix}, \mathbf{x}_1 = \begin{bmatrix} 5.1 \\ 7.0 \\ \vdots \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 5.1 & 3.5 & 1.4 & 0.2 \\ 7.0 & 3.2 & 4.7 & 1.4 \\ & \vdots & & \end{bmatrix}.$$

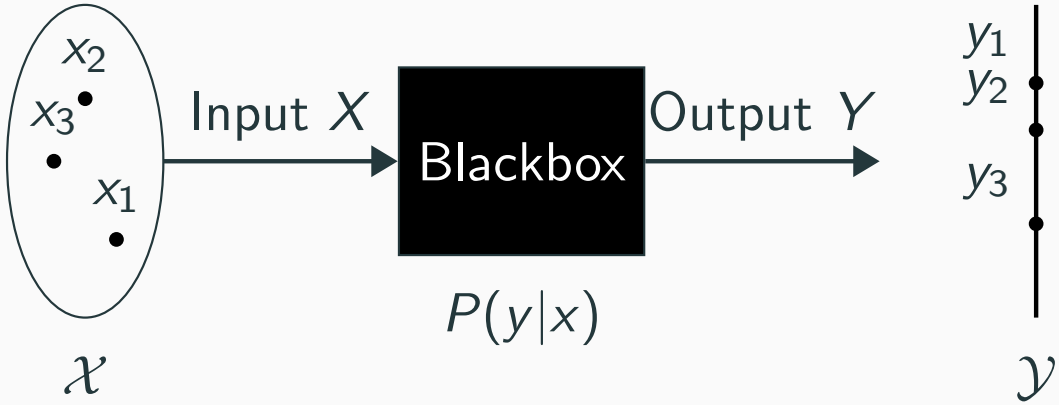
In the above, x_1 is a p-vector, $\mathbf{x}_1$ is an N vector and $\mathbf{X}$ is the $N \times p$ data matrix. Notice that x_1^T is the 1'st row of $\mathbf{X}$.

Inputs and Output



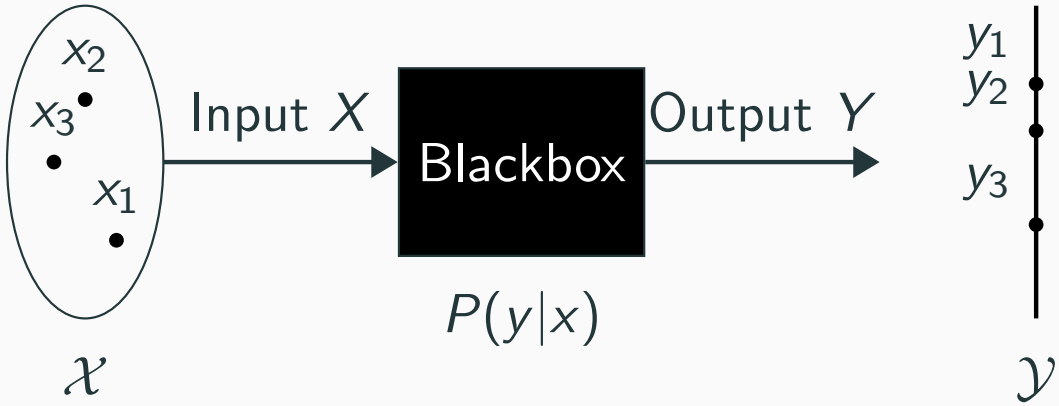
Any fit parameters of predictions will be denoted by a hat $\hat{\theta}$. Assume we have come to a regression function by analyzing the training data (x_i, y_i). The predicted output of the regression function will be denoted $\hat{Y}$.

Inputs and Output



Goal Of Regression Analysis: For any regression problem, we assume that there is an underlying true distribution of labels $P(Y|X)$ (read probability of Y given X) that gives a probability (density) for any data point X in the domain to have label Y. We cant know P, we can only know the data drawn from it, but given that data we want to produce a fit distribution $\hat{P}(Y|X)$ that closely mimics P.

Inputs and Output



Goal Of Regression Analysis: Produce a probability distribution $\hat{P}(Y|X)$ (read probability of Y given X) that gives a probability (density) for any data point x in the domain to have label Y. Usually, $\hat{P}(Y|X)$ is given by a function plus error $\hat{Y} = \hat{f}(X) + \epsilon$.

Inputs and Output

Fitting the regression function: The standard procedure for **fitting** a regression function is start with a **hypothesis class** of functions $\mathcal{H}$ that we will use for our model (e.g. linear functions, 2 layer perceptron networks, etc). We then define a loss function $L(y_i, \hat{f}(x_i)) = L(y_i, \hat{y}_i)$ that gives the error between our prediction $\hat{y}_i$ and the actual label y_i.

Formally, this paradigm of machine learning prescribes finding the $f \in \mathcal{H}$ that minimizes $\sum_i L(y_i, \hat{y}_i)$. The art of modern machine learning is in picking the hypothesis class, finding the minimum loss function within the hypothesis class, picking the loss function, and constructing more robust sets of samples.

First Example: Linear Regression

Linear Regression

The Problem: Given a vector of inputs $X^T = (X_1, \ldots, X_p)$, we want to predict the output Y via a linear model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j \, .$$

The hats here denote fit constants.

If we include a constant variable 1 in X and write $\hat{\beta} = (\beta_0, \ldots, \beta_p)$, we can write

$$\hat{Y} = \hat{\beta}^T X$$

Linear Regression

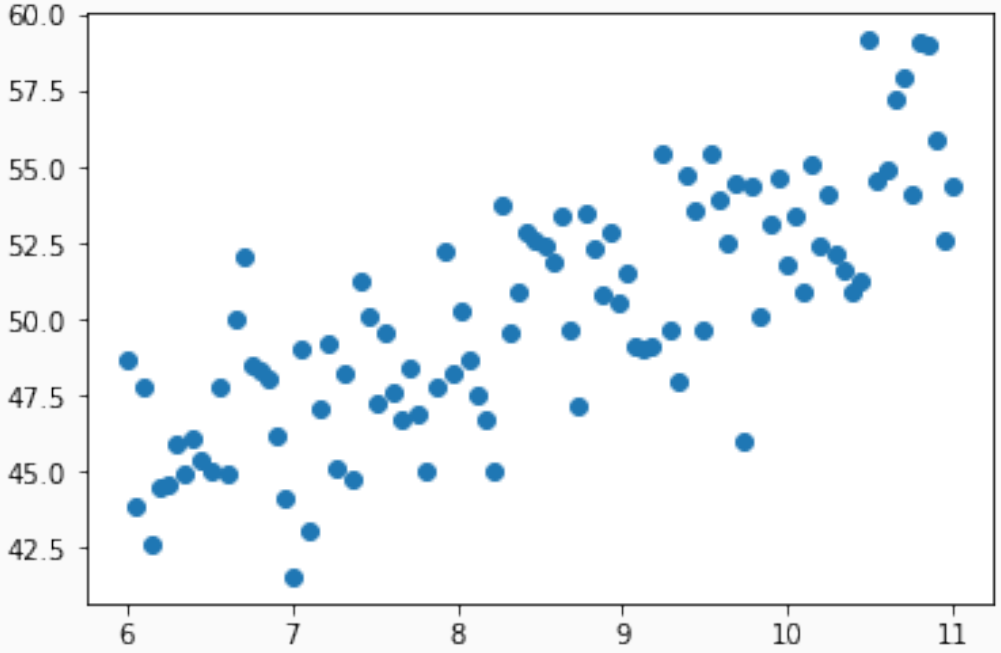
Note that $\hat{Y}$ need not be a scalar: if $\hat{Y}$ is a K vector then $\hat{\beta}$ is an $p \times K$ matrix

$$\begin{bmatrix} \hat{Y}_1 \\ \vdots \\ \hat{Y}_K \end{bmatrix} = \begin{bmatrix} \hat{\beta}_{0,1} \\ \vdots \\ \hat{\beta}_{0,K} \end{bmatrix} + X_1 \begin{bmatrix} \hat{\beta}_{1,1} \\ \vdots \\ \hat{\beta}_{1,K} \end{bmatrix} + \ldots + X_p \begin{bmatrix} \hat{\beta}_{p,1} \\ \vdots \\ \hat{\beta}_{p,K} \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{\beta}_{0,1} & \cdots & \hat{\beta}_{p,0} \\ \vdots & \ddots & \vdots \\ \hat{\beta}_{0,K} & \cdots & \hat{\beta}_{p,K} \end{bmatrix} \begin{bmatrix} 1 \\ X_1 \\ \vdots \\ X_p \end{bmatrix}$$

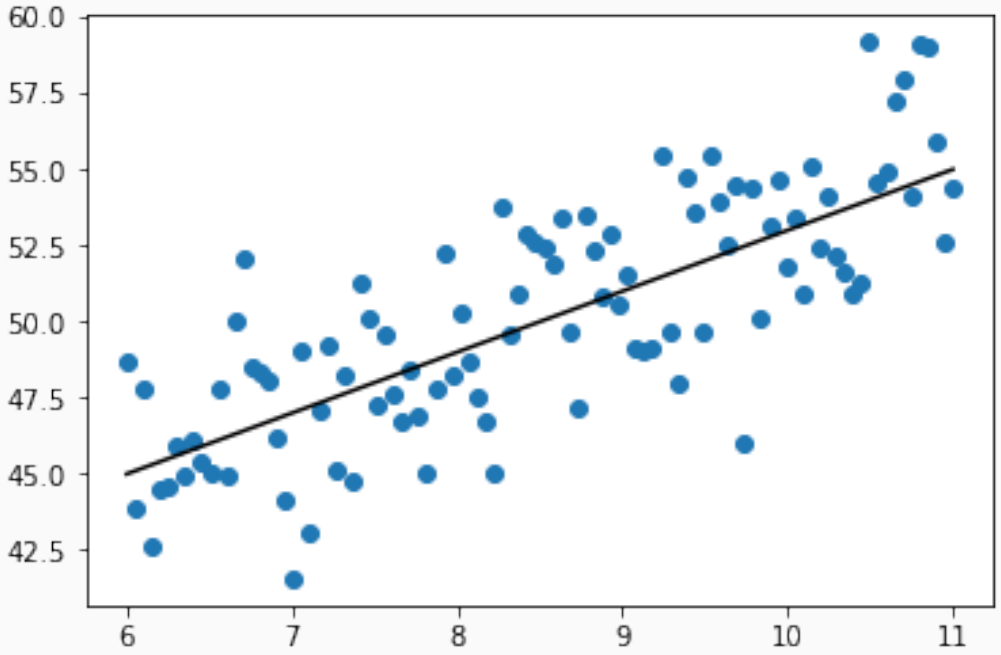
$$\hat{Y} = \hat{\beta}^T X$$

Linear Regression



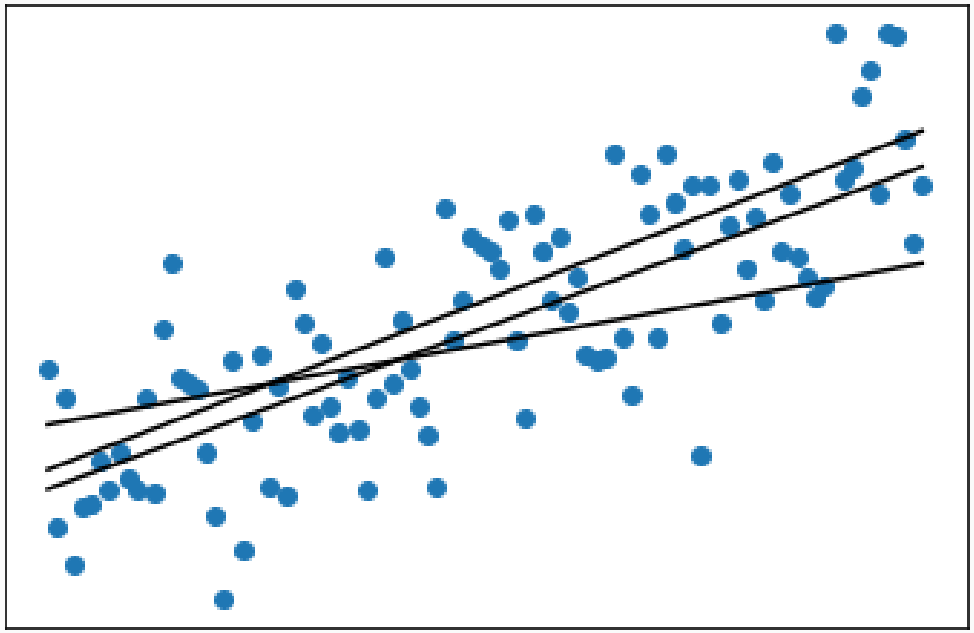
How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

Linear Regression



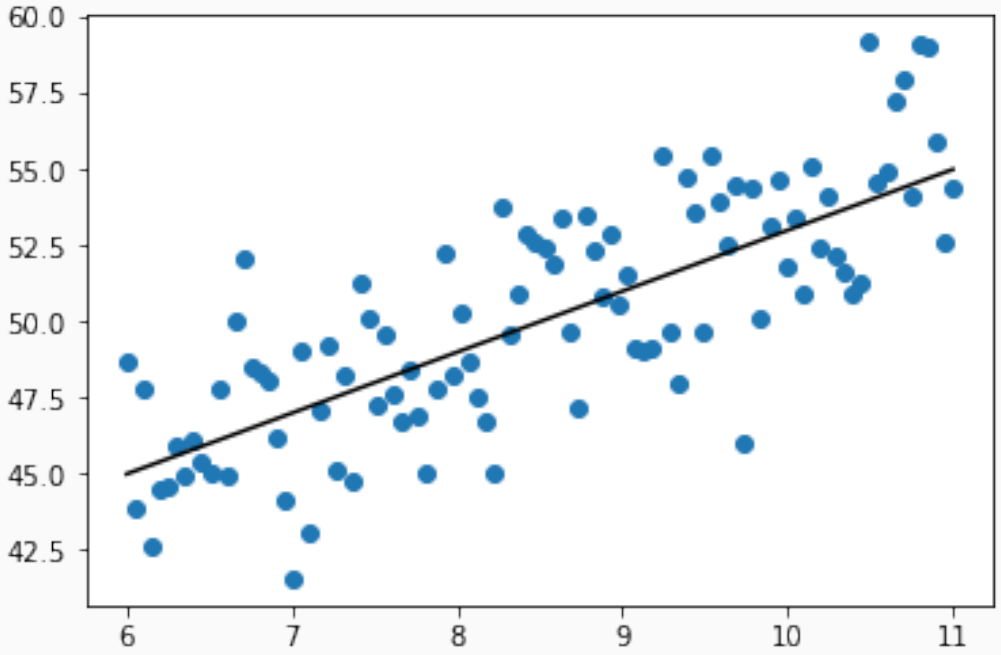
How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

Linear Regression



How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

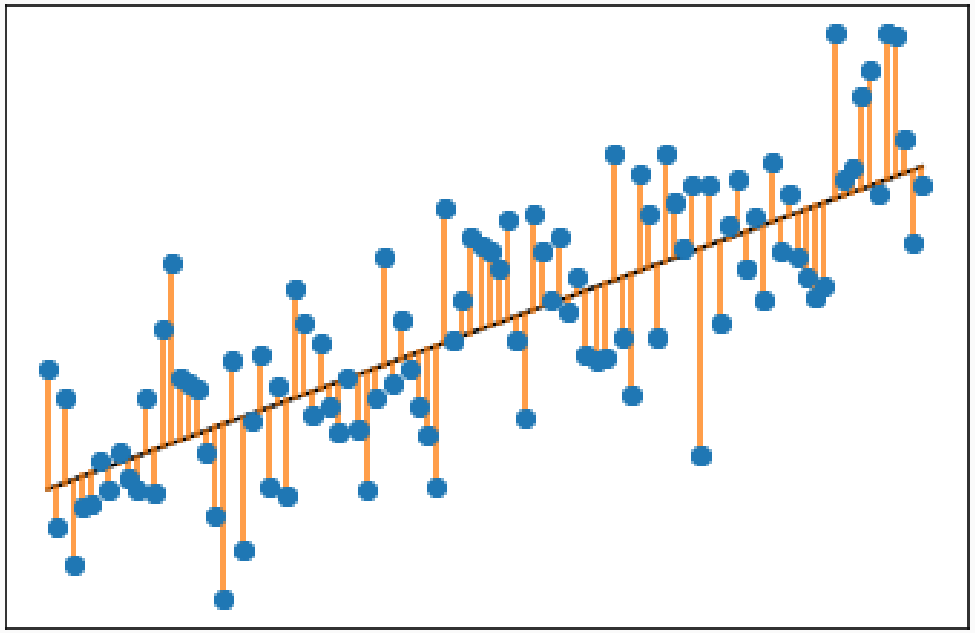
Linear Regression



There are many ways. The most popular is by minimizing the **residual sum of squares** over the training set

$$\text{RSS}(\beta) = \sum_{i=1}^{N} ||y_i - \beta^T x_i||^2 .$$

Linear Regression



There are many ways. The most popular is by minimizing the **residual sum of squares** over the training set. If Y is a scalar,

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta^T x_i)^2 \,.$$

With a little bit of work, we can write

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta^T x_i)^2$$

in matrix form as

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)\,.$$

Proof: First,

$$\mathbf{y} - \mathbf{X}\beta = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} -x_1- \\ \vdots \\ -x_N- \end{bmatrix} \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} y_1 - \beta^T x_1 \\ \vdots \\ y_N - \beta^T x_N \end{bmatrix}\,.$$

Then

$$(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) = \begin{bmatrix} y_1 - \beta^T x_1 & \dots & y_N - \beta^T x_N \end{bmatrix} \begin{bmatrix} y_1 - \beta^T x_1 \\ \vdots \\ y_N - \beta^T x_N \end{bmatrix}$$

$$= \sum_{i=1}^{N} (y_i - \beta^T x_i)^2$$

$$= \mathrm{RSS}(\beta)\,,$$

as claimed.

Linear Regression

Minimizing RSS(β) becomes a problem of calculus. The minimum will occur when the partial derivatives are 0:

$$0 = \frac{\partial}{\partial \beta_i} \text{RSS}(\beta)\,.$$

Homework: Show that the $p \times K$ derivatives

$$\frac{\partial}{\partial \beta} \text{RSS}(\beta) := \frac{\partial}{\partial \beta_{ij}} \text{RSS}(\beta)$$

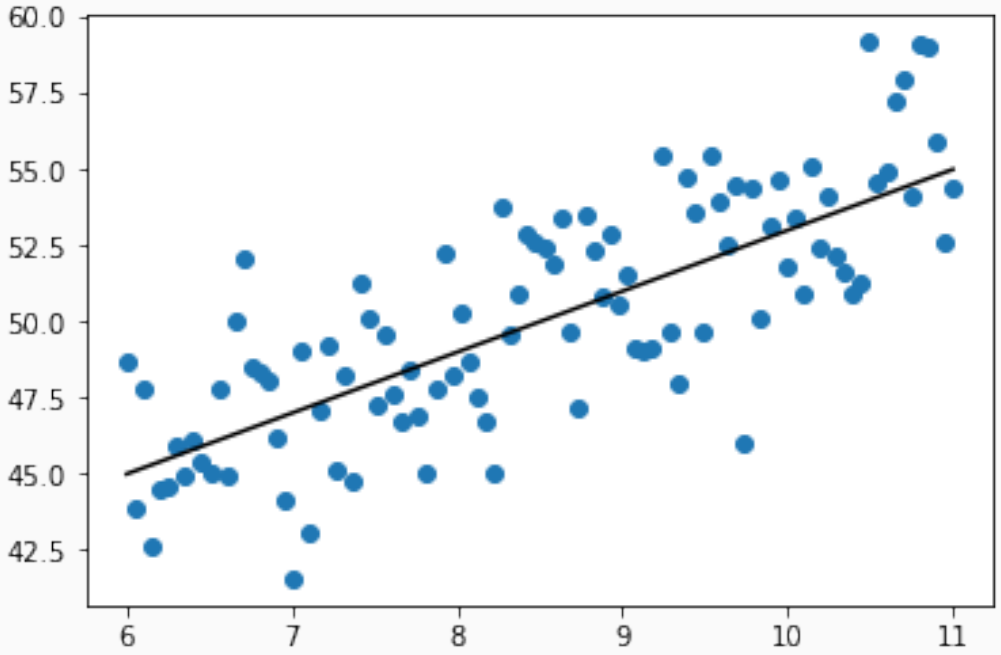
can be written compactly as

$$\frac{\partial}{\partial \beta} \text{RSS}(\beta) = -2\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)\,.$$

Observe that this implies that $\frac{\partial}{\partial \beta} \text{RSS}(\beta) = 0$ when

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0\,.$$

Linear Regression



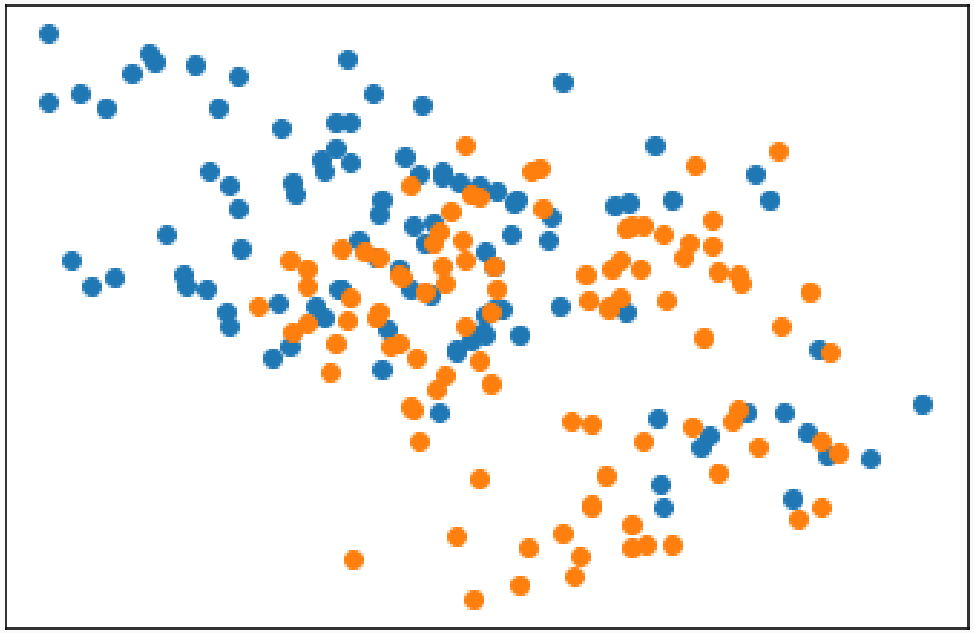
Solving $\mathbf{X}^T(\mathbf{y}-\mathbf{X}\beta) = 0$ for β yields an explicit equation for the parameters

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\,.$$

It doesn't seem to take much data to get a decent answer here, but we do have to invert a matrix. The solution is also relatively **stable** - it would take the addition of many more data points to dramatically change it.

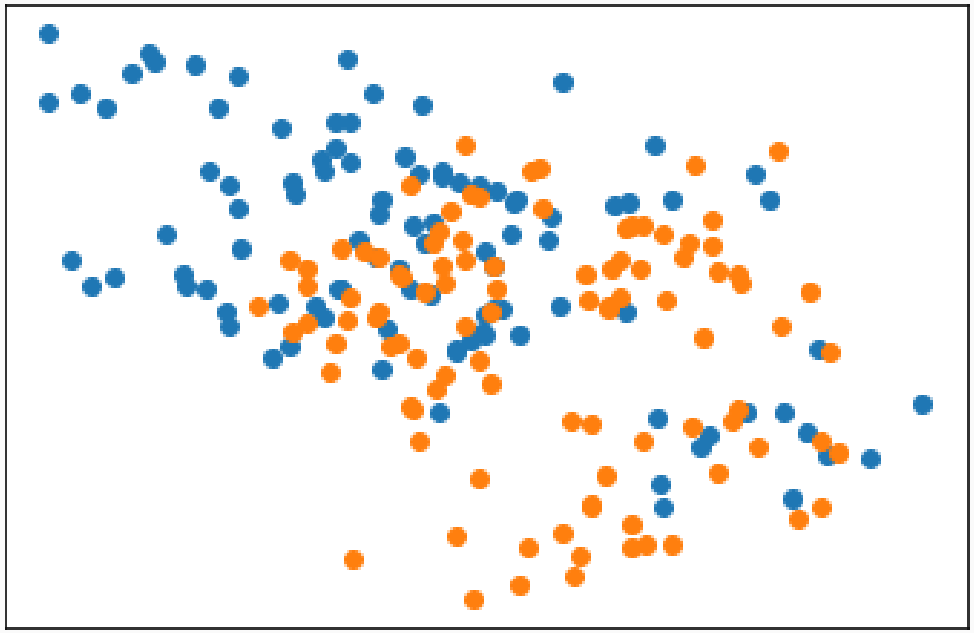
Second Example: Binary Classification

Binary Classification



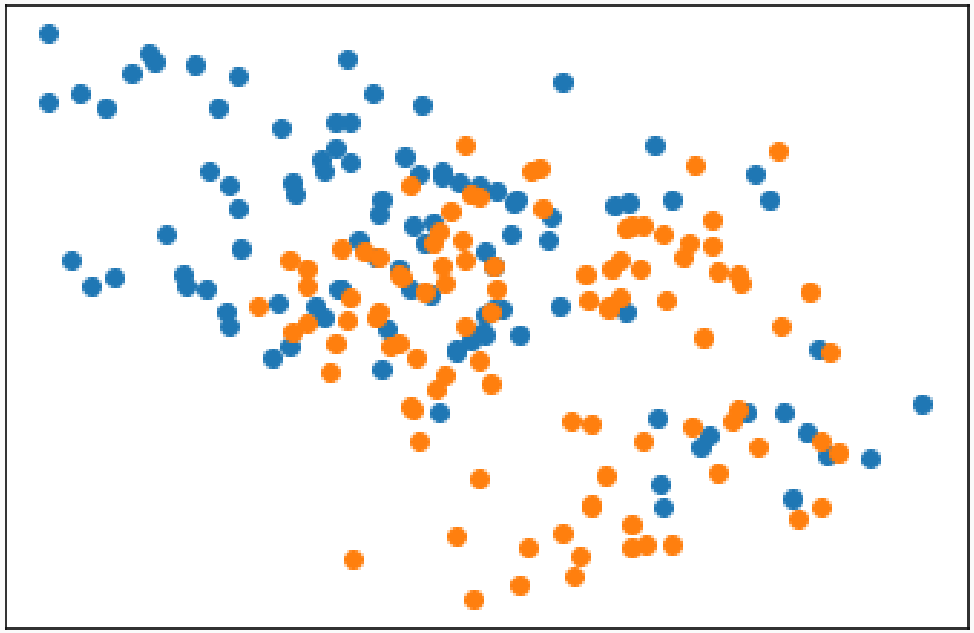
We have here a scatter plot of two colored training data in $\mathcal{X} = \mathbb{R}^2$. The inputs are X_1 and X_2 and the outputs Y take values in $\mathcal{G} = \{\text{Orange}, \text{Blue}\}$.

Binary Classification



For any point (X_1, X_2) we want to produce a probability $\hat{Y}$ that the point is labeled "Orange" and probability $1 - \hat{Y}$ that the point is labeled "Blue."

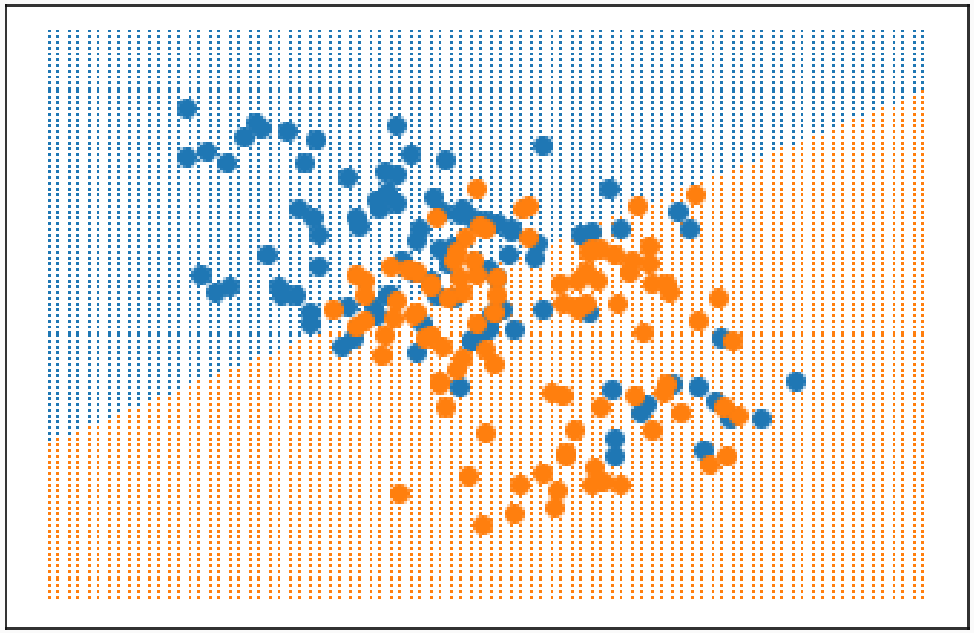
Binary Classification



In the binary case, we may minimize RSS(β) with $Y \in \{0, 1\}$. We assign a color to (X_1, X_2) by selecting

$$\hat{G} = \begin{cases} \text{Orange} & \text{if } \hat{Y} > .5 \\ \text{Blue} & \text{if } \hat{Y} \leq .5 \end{cases}$$

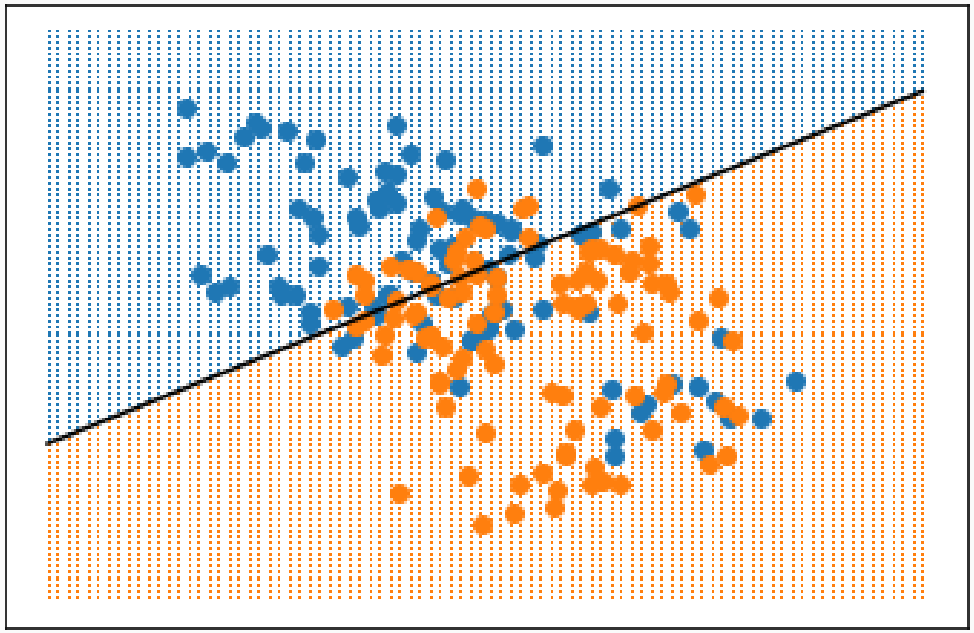
Binary Classification



In the binary case, we may minimize RSS(β) with $Y \in \{0, 1\}$. We assign a color to (X_1, X_2) by selecting

$$\hat{G} = \begin{cases} \text{Orange} & \text{if } \hat{Y} > .5 \\ \text{Blue} & \text{if } \hat{Y} \leq .5 \end{cases}$$

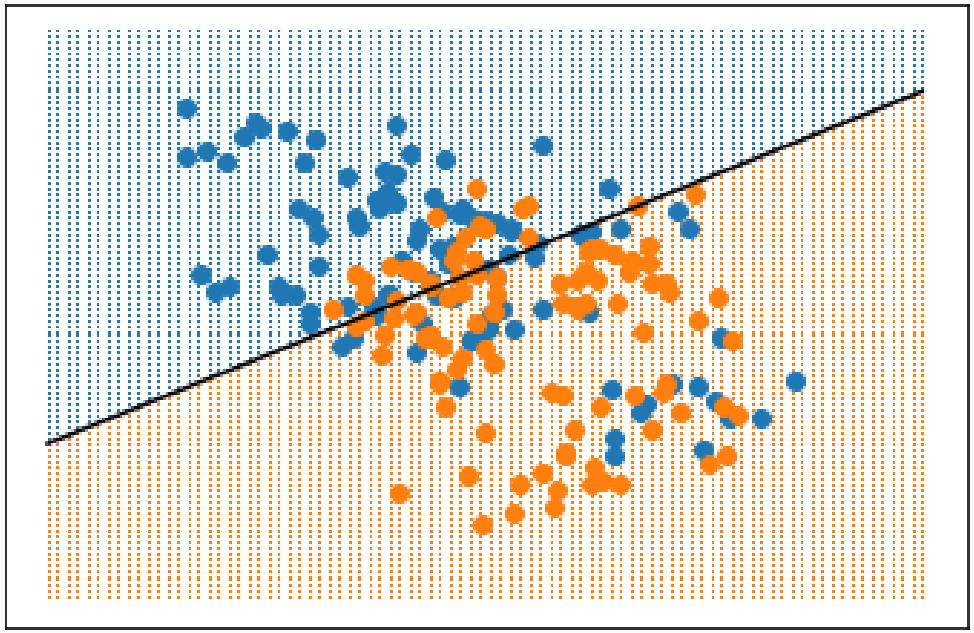
Binary Classification



In this case, the two classes are separated by a linear **decision boundary** $\{x : \hat{\beta} x^T = .5\}$.

Question: What do you think of this model? How could it be improved? Can we minimize the apparent errors?

Binary Classification

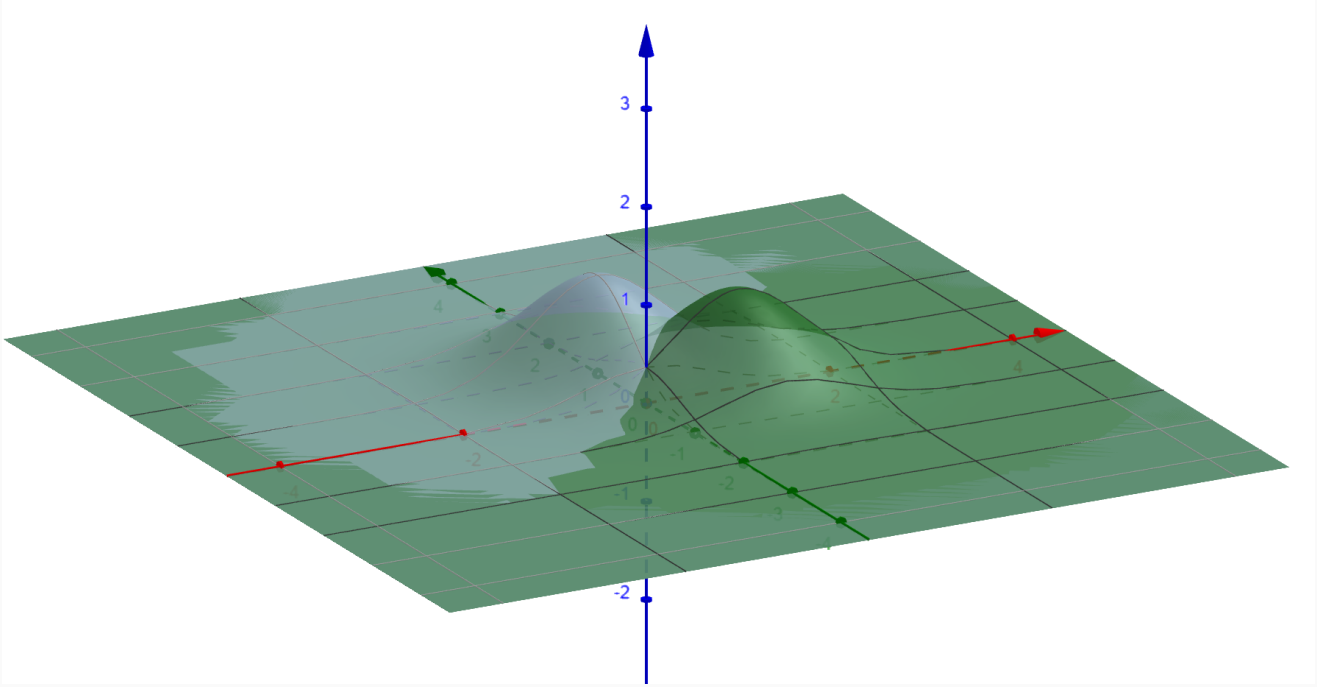


The theoretical accuracy of the linear classifier will depend on the mechanism of data generation. Consider the following two cases:

Scenario 1: The data was generated from a pair of multivariate normal distributions with different means.

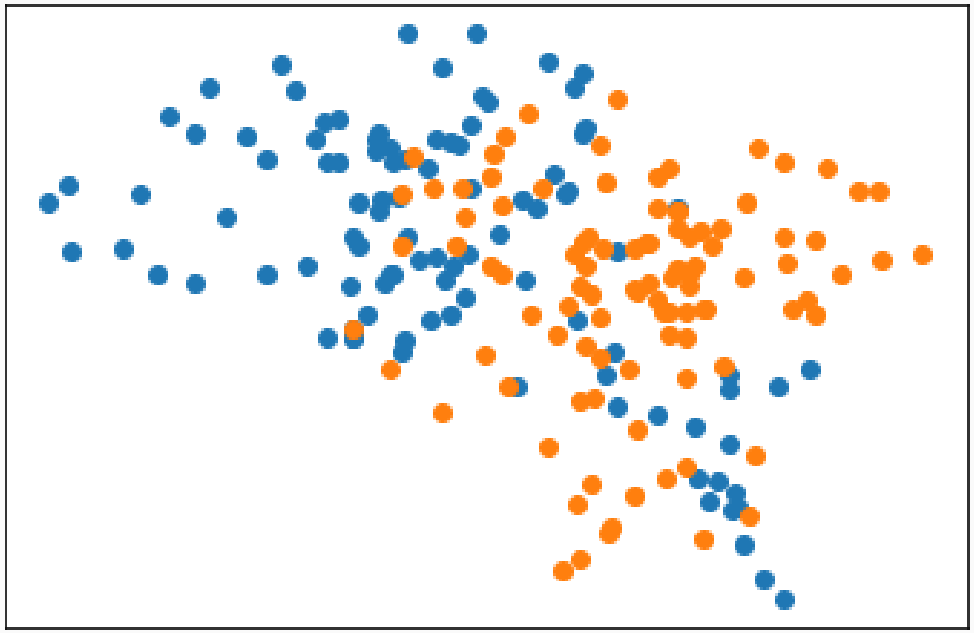
Scenario 2: The data was generated from many low variance distributions with normally distributed means.

Binary Classification



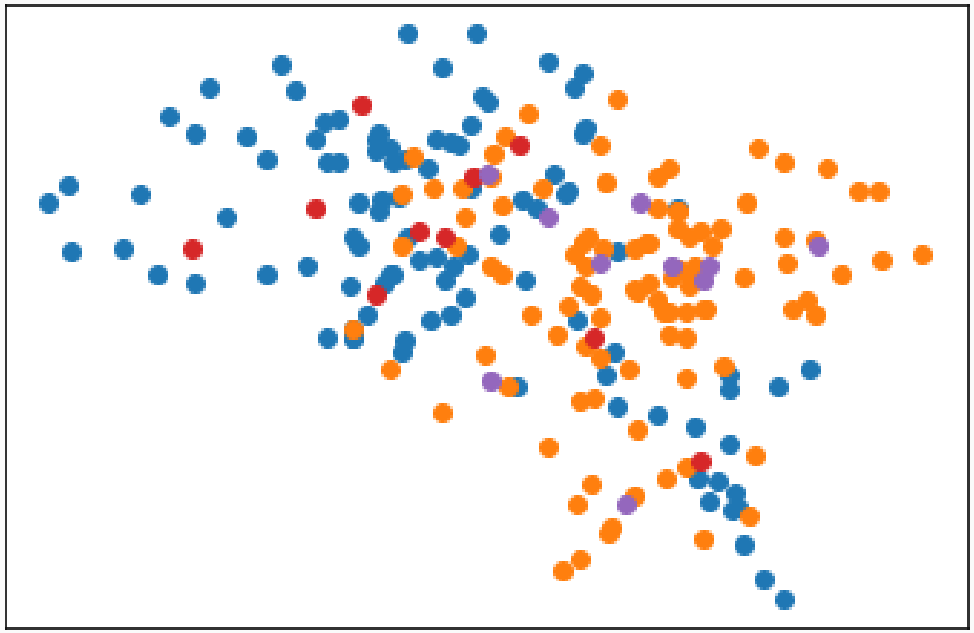
Scenario 1: If data was generated from a pair of multivariate normal distributions with different means, the linear classifier might be the best we can do.

Binary Classification



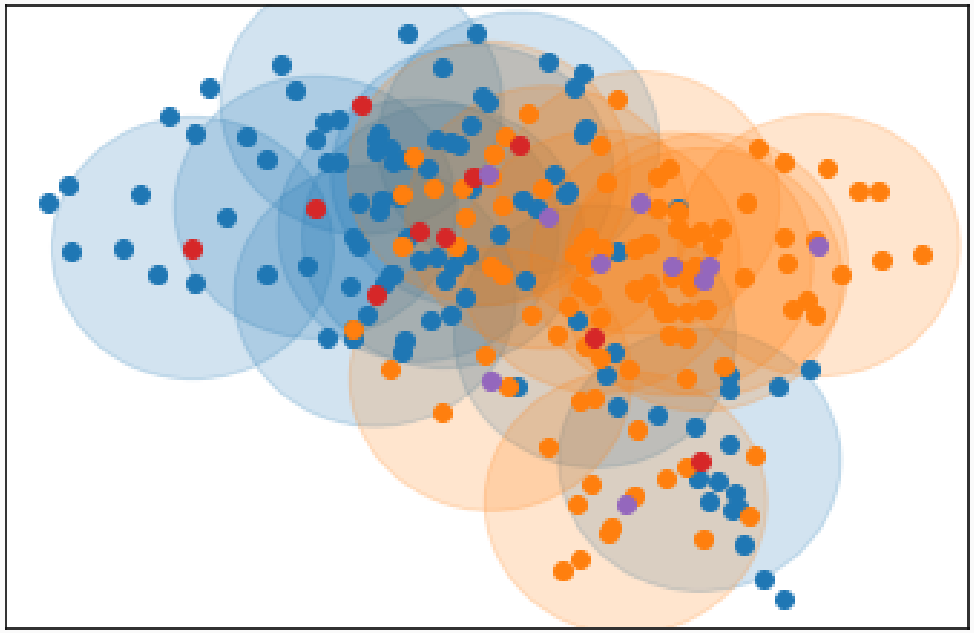
Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier requires a much more nuanced approach.

Binary Classification



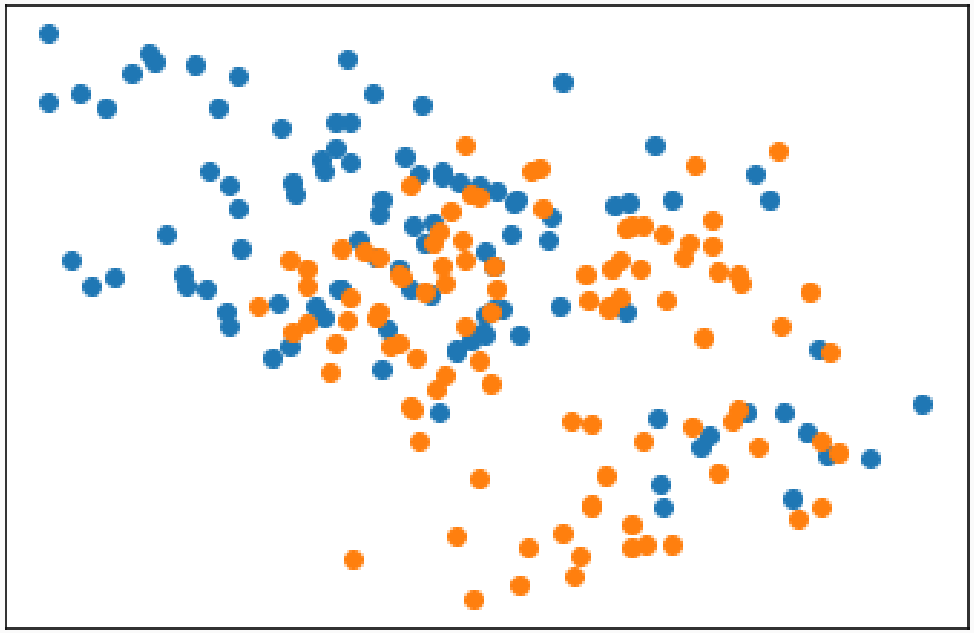
Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier requires a much more nuanced approach.

Binary Classification



Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier would have a much more complicated boundary than a simple line, and may not be connected.

Binary Classification: Nearest Neighbors

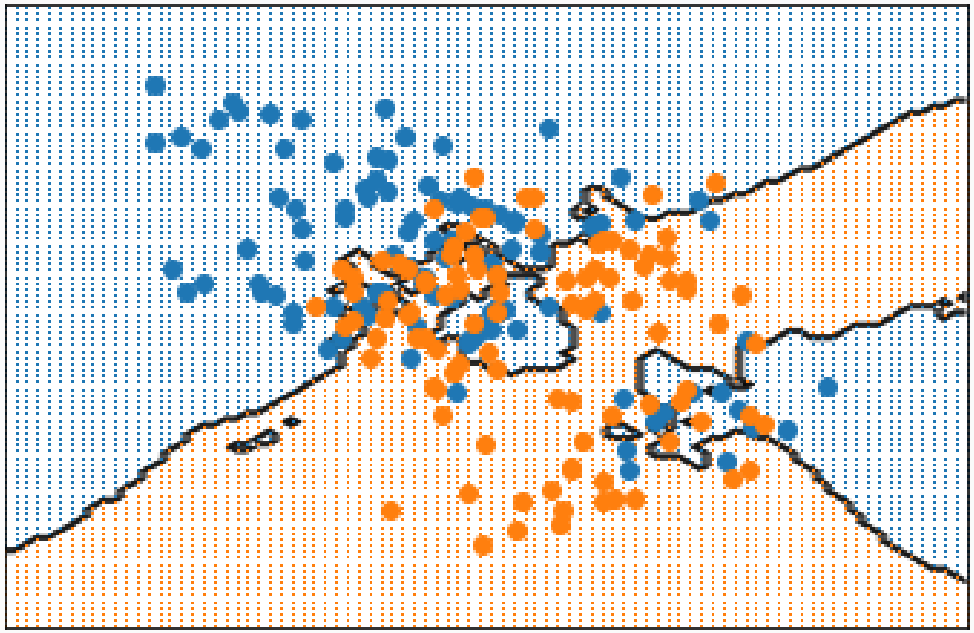


One way to complicate the boundary is the **k-nearest neighbors** method. For any pair of input values X, we determine a labeling by

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \,,$$

where $N_k(x)$ are the k points in the training set closest to x.

Binary Classification: Nearest Neighbors

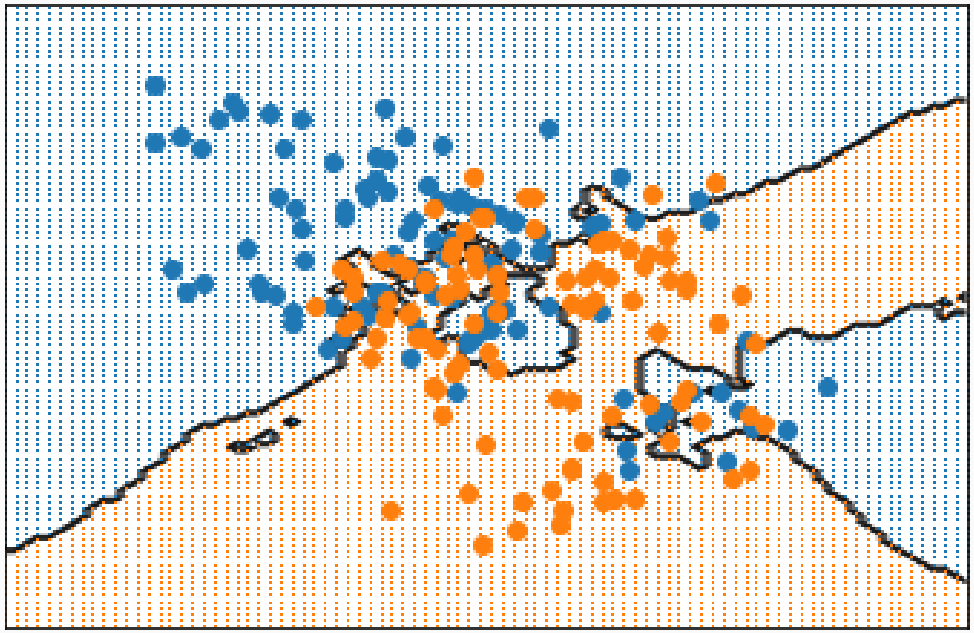


One way to complicate the boundary is the **k-nearest neighbors** method. For any pair of input values X, we determine a labeling by

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \,,$$

where $N_k(x)$ are the k points in the training set closest to x.

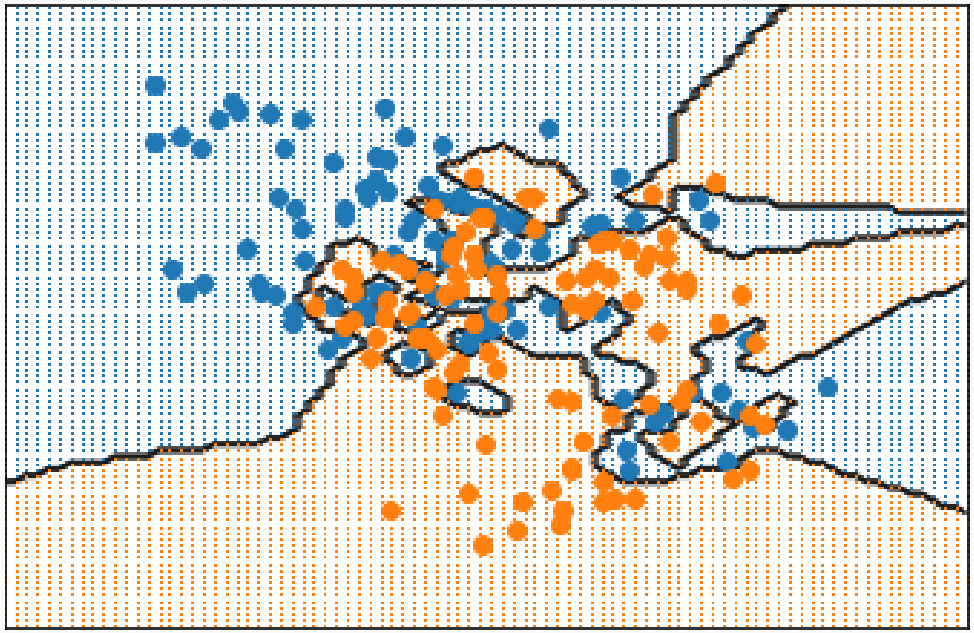
Binary Classification: Nearest Neighbors



Notice that the boundary is much more jagged in this case, but we also have fewer misclassifications of the training data. Such misclassifications are can be expressed as the **training error**.

The training error is roughly an increasing function of k. When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50% (assuming equal number of each label).

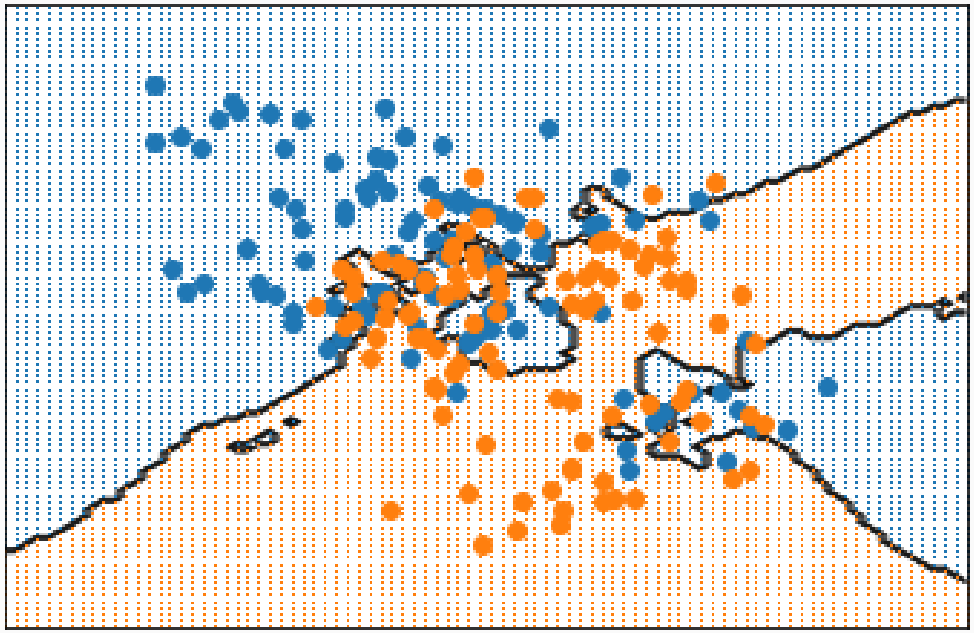
Binary Classification: Nearest Neighbors



$\mathbf{k = 1}$

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

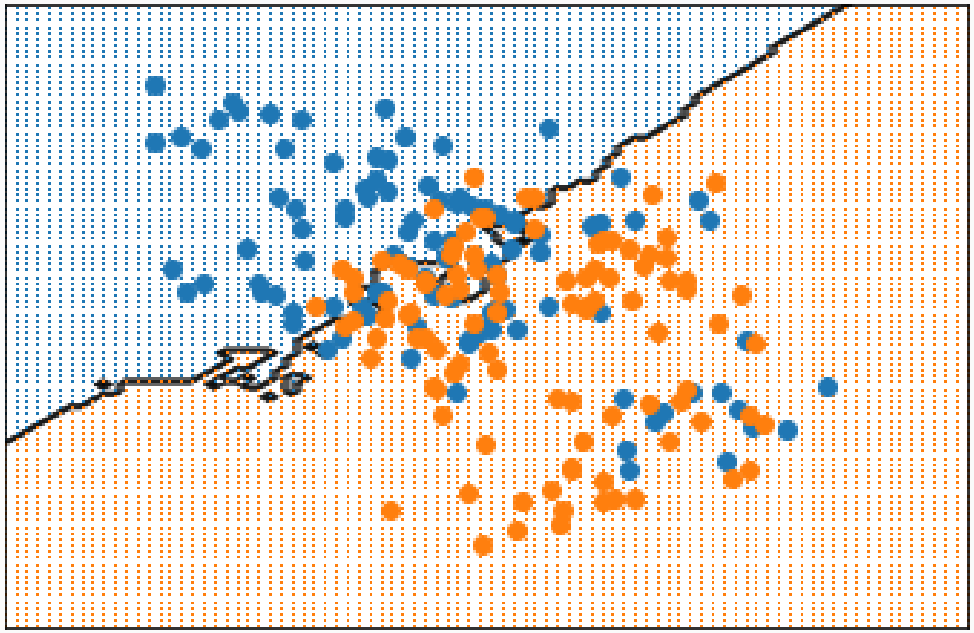
Binary Classification: Nearest Neighbors



k = 10

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

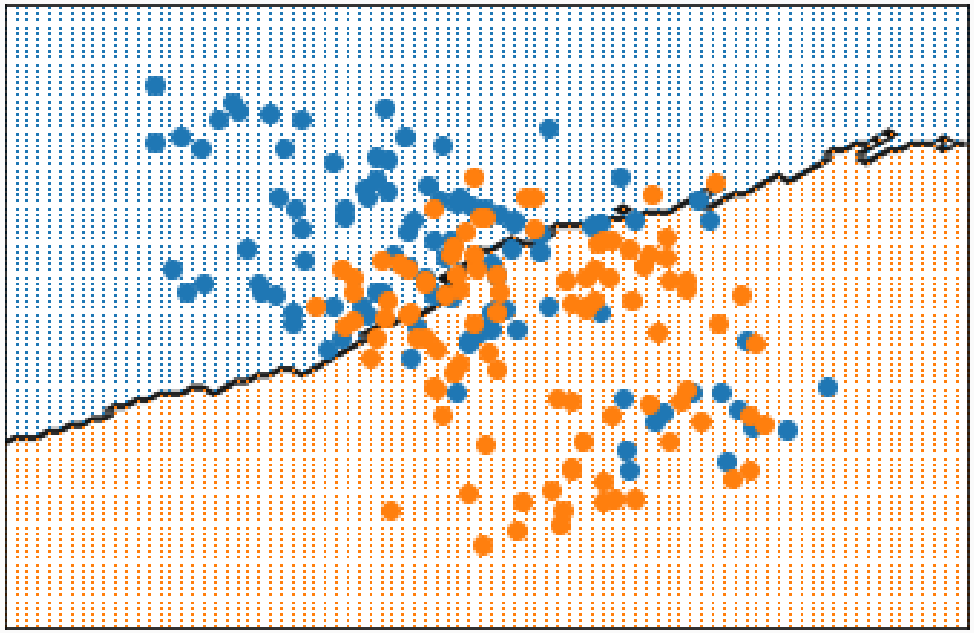
Binary Classification: Nearest Neighbors



k = 50

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

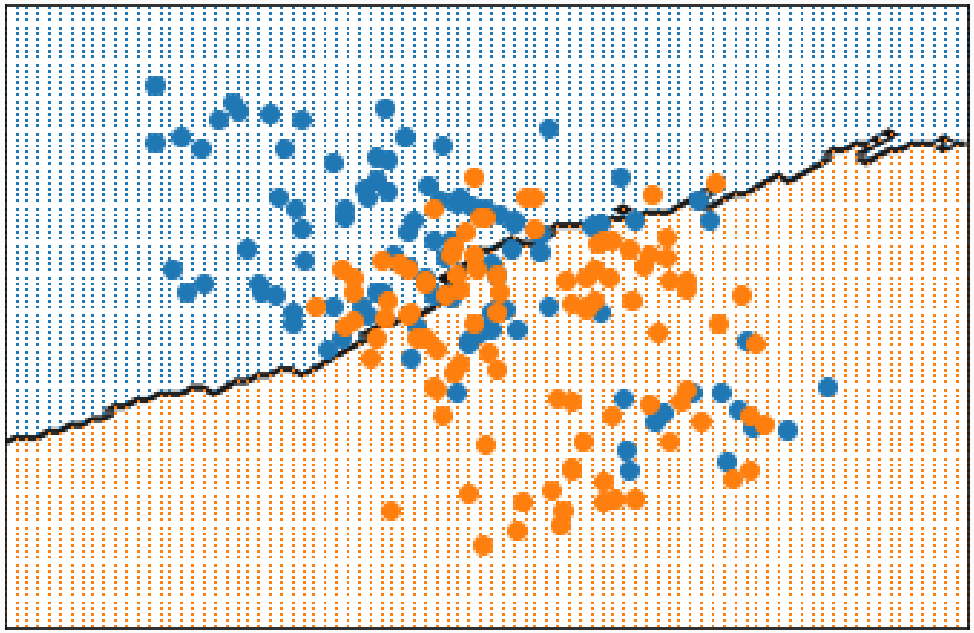
Binary Classification: Nearest Neighbors



$\mathbf{k = 100}$

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

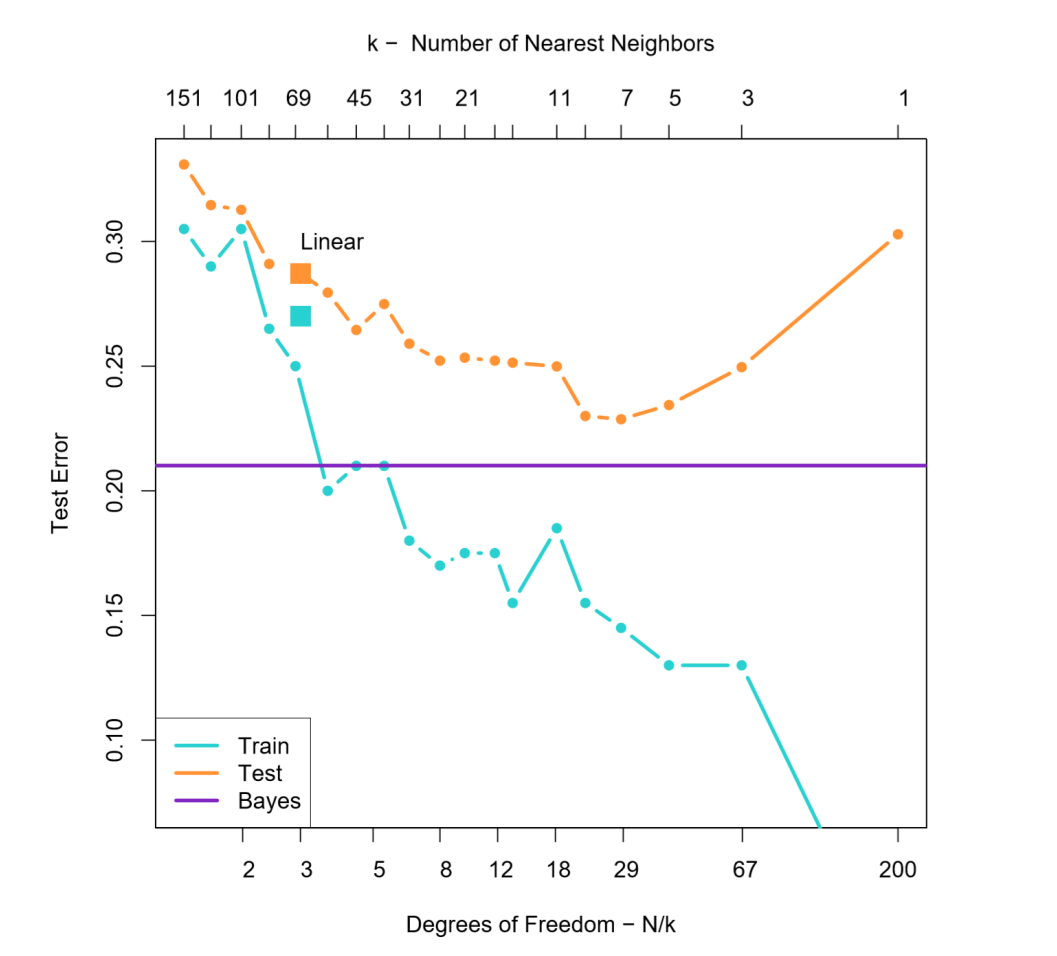
Binary Classification: Nearest Neighbors



$\mathbf{k = 100}$

Note: It appears that k-nearest neighbors has a single parameter k, but we can't fit this parameter using lease squares since we will always just get $k = 1$.

Binary Classification: Comparison



Comparing the linear model to the k-nearest neighbors, we see that while the training error decreases as N/k increases, the true error is relatively high on both ends.

Course Structure and Policies

Course Texts

This course will follow a pair a text books:

> **High level:** *The Elements of Statstical Learning*. Trevor Hastie Robert Tibshirani, Jerome Friedman.
>
> **Low level:** *Understanding Machine Learning: From Theory to Algorithms*. Shai Shalev-Shwartz, Shai Ben-David.

There is also a practicum book which is worth having if you're interested in machine learning:

> **Implementation:** *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Aurélien Géron.

Society and impact:

> **Implementation:** *Weapons of math destruction*. Cathy O'Neil.

Course Policies

The course grade will be divided up as follows:

30% Homework

30% Labs

40% Final Project

Homework

Homeworks will be due Tuesday in class in weeks 3, 5 and 7. You are free to discuss homework with me in office hours. You may work in study groups, but all work turned in must be your own. Homework will be partially graded on presentation, and extra points will be awarded for homework completed in latex.

Labs

Labs will use Python and will be distributed as Jupyter Notebooks. They will be turned in through Blackboard. There will be roughly 6 labs beginning in week 4. The labs will be partially in class, and should take no longer than the allotted class time in theory, but extra credit will be given for going above and beyond, both in presentation and in results.

Final Project

Project

The final project will consist of a project report (roughly 5 pages) and presentation (roughly 10 minutes).

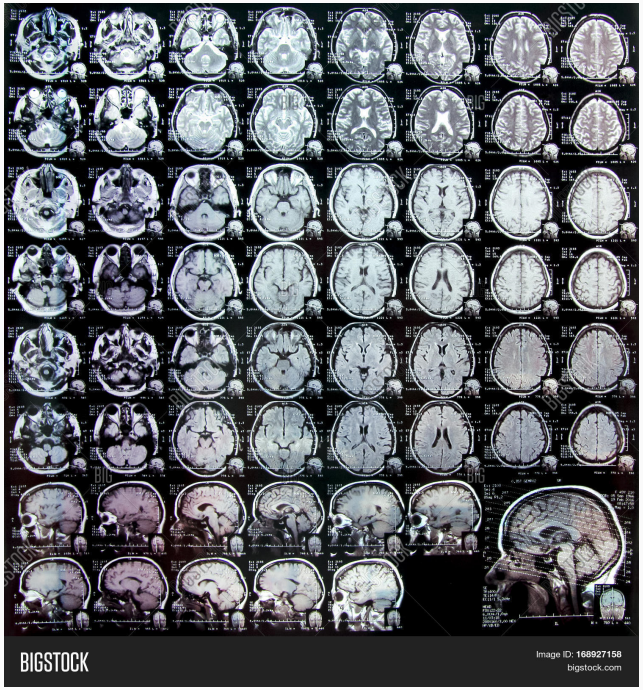
Project groups should contain 2-4 people.

Projects can be of one of three forms:

- A computational analysis of a data set using sufficiently complicated or novel techniques from this course.
- A theoretical presentation of a topic not covered in this course with a case study.
- Machine learning project from external source (industry, PhD work, etc)

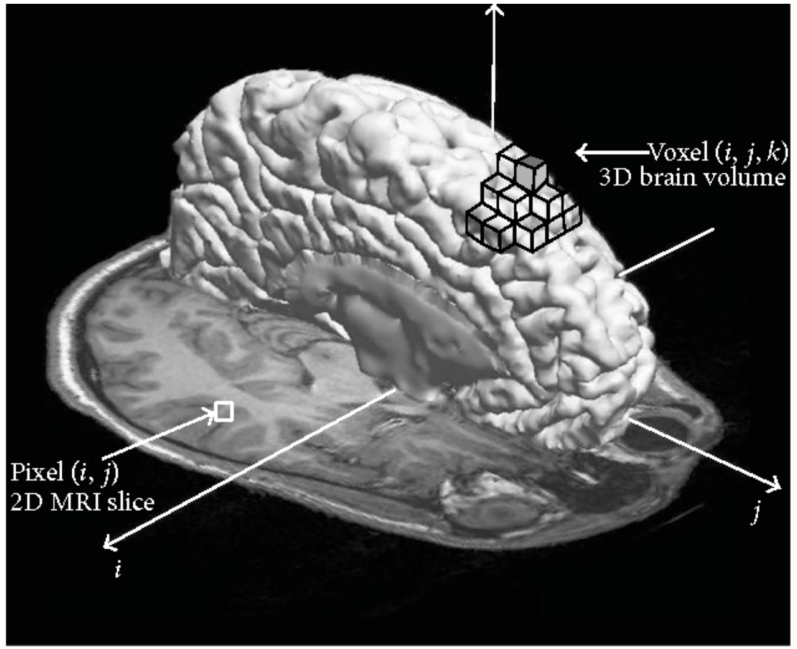
I am more than happy to discuss possible projects in any of these categories with you.

Automatic segmentation tool for medical imaging



This project would involve taking stacks of 2d MRI data and segmenting it to pull out 3d features. The goal of the project is to eventually register the 3d images with depth scans (video) of patients heads, so that a surgeon can see a patients head with the volumetric features from the MRI super imposed.

Automatic segmentation tool for medical imaging



Training data will be a set of MRI scans from patients in one of three stages of dementia. The task will be to train a volumetric convolutional net that can classify a scan into the stages of dementia. The OASIS 1 dataset contains 416 volumetric MRI images at 12 MB each, or around 5 Gb of data.

Automatic segmentation tool for medical imaging

Initially, it will be a simple classification project. Using the OASIS (Open Axis Series of Imaging Studies) 1 MRI dataset, you will try to classify MRI scans as coming from the brain of a patient diagnosed with dementia.

I will provide a 2d sampling of data containing around 600 images, your first attempts at classification will be from that dataset. As the semester goes on, you should expect to use the tools presented to try to improve your fit, and eventually to use a full 3d convolutional neural network to fit the data.

There is a lot of room in this project to pick up soft skills like git, AWS, and Google Cloud. Groups are recommended to heavily delegate and work together across groups.

The 2D images dataset has been posted on blackboard and will be incorporated into some of the labs.

Project

Rough timeline:

- Group Selection: January 27
- Project Proposal Deadline: February 3
- Progress Report: February 24
- First Draft: March 16
- Final Draft: April 6
- Presentations: April 13 - 24.

These may change depending on the loss function.